68' MICRO JOURNAL

Australia A $4.75 | New Zealand NZ $ 6.50
Singapore S $9.45 | Hong Kong H $23.50
Malaysia M $9.45 | Sweden 30:-SEK

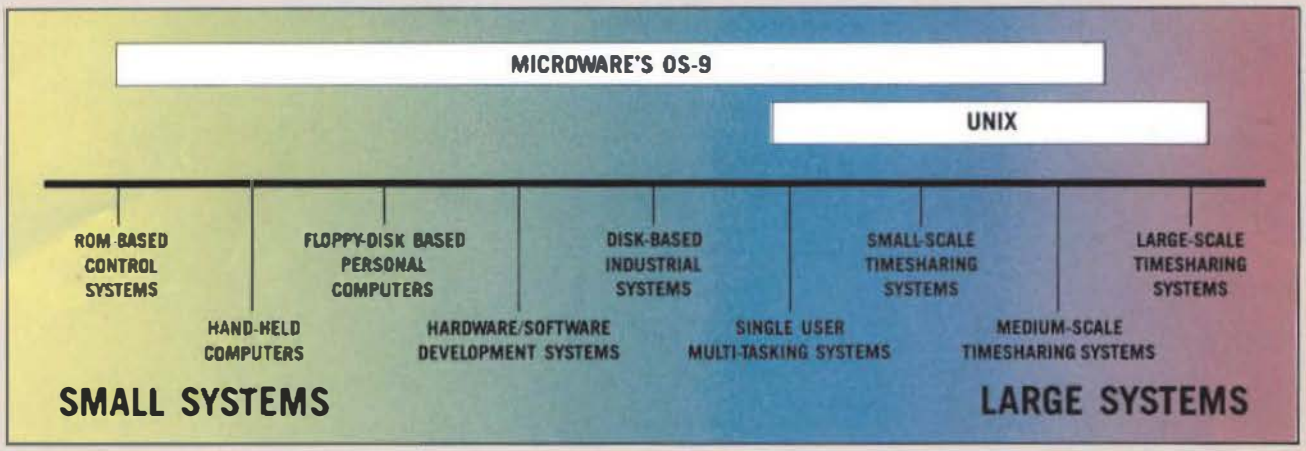
$2.95 USA

VOLUME VIII ISSUE II • Devoted to the 68XX User • February 1986

"Small Computers Doing Big Things"

0 74470 12810 02

Only Microware's OS-9 Operating System Covers the Entire 68000 Spectrum



Is complicated software and expensive hardware keeping you back from Unix? Look into OS-9, the operating system from Microware that gives 68000 systems a Unix-style environment with much less overhead and complexity.

OS-9 is versatile, inexpensive, and delive s outstanding performance on any size system. The OS-9 executive is much smaller and far more efficient than Unix because it's written in fast, compact assembly language, making it ideal for critical real-time applications. OS-9 can run on a broad range of 8 to 32 bit systems based on the 68000 or 6809 family MPUs from ROM-based industrial controllers up to large multiuser systems.

OS-9'S OUTSTANDING C COMPILER IS YOUR BRIDGE TO UNIX

Microware's C compiler technology is another OS-9 advantage. The compiler produces extremely fast, compact, and ROMable code. You can easily develop and port system or application software back and forth to standard Unix systems. Cross-compiler versions for VAX and PDP-11 make coordinated Unix/OS-9 software development a pleasure.

SUPPORT FOR MODULAR SOFTWARE — AN OS-9 EXCLUSIVE

Comprehensive support for modular software puts OS-9 a generation ahead of other operating systems. It multiplies programmer productivity and memory efficiency. Application software can be built from individually testable software modules including standard "library" modules. The modular structure lets you customize and reconfigure OS-9 for specific hardware easily and quickly.

A SYSTEM WITH A PROVEN TRACK RECORD

Once an underground classic, OS-9 is now a solid hit. Since 1980 OS-9 has been ported to over a hundred 6809 and 68000 systems under license to some of the biggest names in the business. OS-9 has been imbedded in numerous consumer, industrial, and OEM products, and is supported by many independent software suppliers.

Key OS-9 Features At A Glance

- Compact (16K) ROMable executive written in assembly language
- User "shell" and complete utility set written in C
- C-source code level compatibility with Unix
- Full Multitasking/multiuser capabilities
- Modular design - extremely easy to adapt, modify, or expand
- Unix-type tree structured file system
- Rugged "crash-proof" file structure with record locking
- Works well with floppy disk or ROM-based systems
- Uses hardware or software memory management
- High performance C, Pascal, Basic and Cobol compilers

microware® OS-9

AUTHORIZED MICROWARE DISTRIBUTORS

AUSTRALIA
MICROPROCESSOR CONSULTANTS
16 Bandera Avenue
Wagga Wagga 2650
NSW Australia
phone: 616-931-2331

ENGLAND
VIVAWAY LTD.
36-38 John Street
Luton, Bedfordshire
England LU1 2JE
phone: (0582) 423425
telex: 825115

JAPAN
MICROWARE JAPAN LTD.
3-8-9 Baraki, Ichikawa
Chiba 272-01, Japan
phone: 0473 (28) 4493
telex: 781-299-3122

SWEDEN
MICROMASTER SCANDINAVIAN AB
Sd Persgatan 7
Box 1309
S751-43 Uppsala
Sweden
phone: 018-138595
telex: 76129

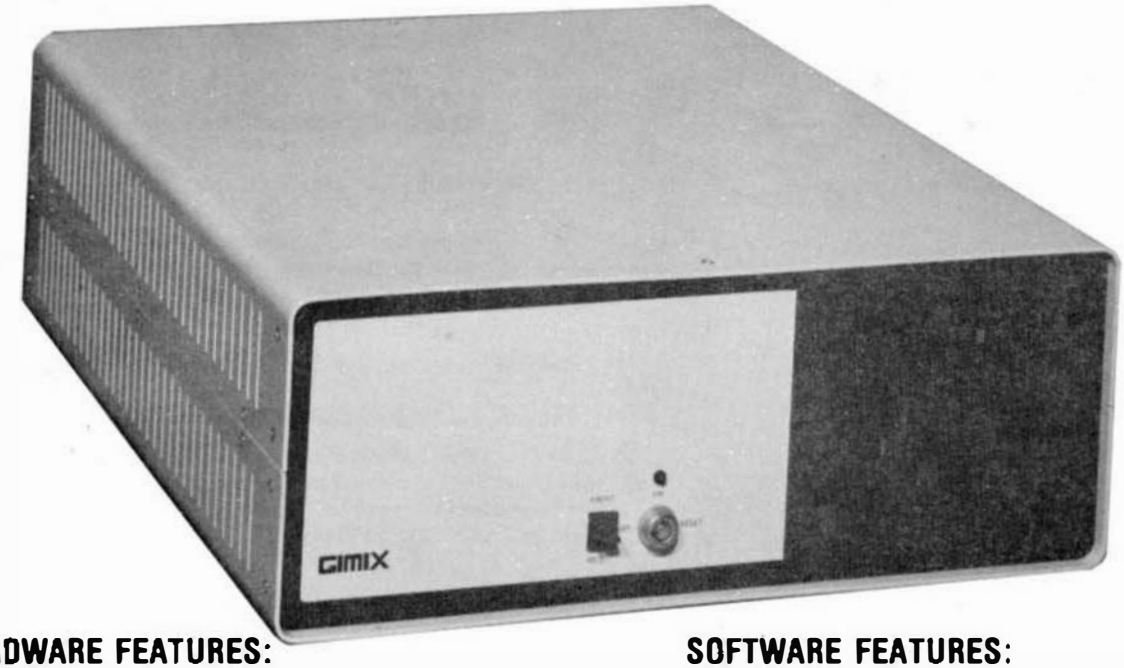
SWITZERLAND
ELSOFT AG
Bankstrasse 9
5432 Neuenhof
Switzerland
phone: (41) 056-862724
telex: 57136

USA
MICROWARE SYSTEMS CORPORATION
1866 NW 114th Street
Des Moines, Iowa 50322
USA
phone: 515-224-1929
telex: 910-520-2535
FAX: 515-224-1352

WEST GERMANY
DR. KEIL, GMBH
Porphystrasse 15
D-6905 Schriesheim
West Germany
phone: (0 62 03) 67 41
telex: 465025

GMX 68020 DEVELOPMENT SYSTEM

A Multi-user, Multi-tasking software development system for use with all 68000 family processors.



HARDWARE FEATURES:

- The GMX-020 CPU board has: the MC68020 32-bit processor, a 4K Byte no wait-state Instruction cache, high-speed MMU, and a full-featured hardware time of day clock/calendar with battery back-up. It also provides for an optional 68881 floating point co-processor.
- 1 Megabyte of high speed static RAM.
- Intelligent Serial and Parallel I/O Processor boards significantly reduce system overhead by handling routine I/O functions. This frees up the host CPU for running user programs. The result is a speed up of system performance and allows all terminals to run at up to 19.2K baud.
- The system hardware will support up to 39 terminals.
- Powered by a constant voltage ferro-resonant power supply that insures proper system operation under adverse AC power Input conditions.
- DMA hard disk interface and DMA double density floppy disk controller are used for data transfers at full bus speed. The DMA hard disk drive controller provides automatic 22-bit burst data error detection and 11-bit burst error correction.
- A selection of hard disk drives with capacities from 19 to 85 Megabytes, removeable pack hard disk drives, streaming tape drives, and floppy disk drives Is available.

SOFTWARE FEATURES:

The UniFLEX VM Operating System is a demand-paged, virtual memory operating system written in 68020 Assembler code for compactness and efficiency. Any UniFLEX system will run faster than a comparable system written in a higher level language. This is important In such areas as context switching, disk I/O, and system call handling. Other features include:

- compact, efficient Kernel and modules allows handling more users more effectively than UNIX systems, using much less disk space.
- UNIX system V compatibility at the C source code level.
- C Compiler optimized In 68020 code (optional).
- Record locking for shared files.
- Users can share programs in memory.
- Modeled after UNIX systems, with similar commands.
- System accounting facilities.
- Sequential and random file access.
- Maximum record size limited only by the disk size.
- Multiple Level Directories.
- Up to 4 Megabytes of Virtual Memory per user.
- Optional Languages available are: C, BASIC, COBOL, FORTRAN, LISP, PROLOG, SCULPTOR, and ASSEMBLER. In development are ADA, PASCAL, and FORTH.

Included with the UniFLEX Operating System are a Utilities package, editor, relocating assembler, linking loader, and printer spooler. Options inlcude a fast floating point package, library generator, and a sort-merge package.

The GMX version of the MOTOROLA 020 BUG Is included with the system.

GIMIX, Inc., a Chicago based microcomputer company established in 1975, has produced state of the art microcomputer systems based on Motorola 6800 and 6809 microprocessors. GIMIX systems are in use in Industry,, Hospitals, Universities, Research Organiations, and by Software Developers. GIMIX was awarded the prestigious President's "E" Certificate for Exports in 1984.

GIMIX INC. 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX910-221-4055

'68' MICRO JOURNAL

Portions of the text for '68' Micro Journal were prepared using the following furnished Hard/Software:

COMPUTERS – HARDWARE
Southwest Technical Products
219 W. Rhapsody
San Antonio, TX 78216
S.9 - 5/8 DMF Disk - COS1 - 8212W - Sprint 3 Printer

GIMIX Inc.
1337 West 37th Place
Chicago, IL 60609
Super Mainframe - OS9 - FLEX - Assorted Hardware

EDITORS – WORD PROCESSORS
Technical Systems Consultants, Inc.
111 Providence Road
Chapel Hill, NC 27514
FLEX - Editor - Text Processor

Stylo Software Inc.
PO Box 916
Idaho Falls, ID 83402
Stylograph - Mail Merge - Spell

Contents

Vol. VIII, Issue 2 February 1986

Send All Correspondence To:

Computer Publishing Center
68' Micro Journal
5900 Cassandra Smith Rd.
Hixson, TN 37343

Phone (615) 842-4600 or Telex 5106006630

68' Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage Paid ISSN 0194-5025 at Hixson, TN and additional entries. Postmaster: send form 3597 to 68' Micro Journal, POB 849 Hixson, TN 37343.

Subscription Rates

1 Year $24.50 U.S.A., Canada & Mexico Add **$9.50** a Year. Other **Foreign Add $12 a Year** for **Surface, Airmail Add $48 a Year.** Must be in **U.S. currency!!**

Items or Articles For Publication

Articles submitted for publication should include authors name, address, telephone number and date. Articles should be **on either 5 or 8 inch disk** in STYLOGRAPH or TSC Editor format with **3.5 inch column width.** All disks will be returned. Articles submitted **on paper** should be **4.5 inches in width** (including Source Listings) for proper reductions. **Please Use A Dark Ribbon!!** No Blue Ink!!! Single space on 8x11 bond or better grade paper. No hand written articles accepted. Disks should be in FLEX2 6800 or FLEX9 6809 any version or OS-9 any version.

The following TSC Text Processor commands **ONLY** should be used: **.sp** space, **.pp** paragraph, **.fl** fill and **.nf** no fill. Also please **do not format** within the text with **multiple spaces.** We will enter the rest at time of editing.

All STYLOGRAPH commands are acceptable except **.pg** page command. We print edited text files in continous text form.

Letters To The Editor

All letters to the editor should comply with the above requirements and **must be signed.** Letters of "gripes" as well as "praise" are solicited. We reserve the right to reject any submission for lack of "good taste" and we reserve the right to define "good taste".

Advertising Rates

Commercial advertisers please contact 68' Micro Journal advertising department for current rate sheet and requirements.

Classified Advertising

All classified ads must be non-commercial. Minimum of **$9.50 for first 20 words** and .45 per word after 20. All classifieds must be **paid in advance.** No classified ads accepted over the phone.

THE 6800-6809 BOOKS

..HEAR YE.....HEAR

OS-9™ User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibbles **OS9 USER NOTES**

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled Operating Programs. The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing your OWN more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK $9.95

Typeset -- w/ Source Listings
(3-Hole Punched; 8 x 11)
Deluxe Binder - - - - - - - - - - $5.50

All Source Listings on Disk

1-8" SS, SD Disk - - - - $14.95
2-5" SS, DD Disks - - - $24.95

Shipping & Handling: $3.50 per **Book**, $2.50 per **Disk** set
* All Currency in U.S. Dollars
Foreign Orders Add $4.50 S/H (Surface)
or $7.00 S/H (Air Mail)
If paying by check - Please allow 4-6 weeks delivery

FLEX™ USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC

NOTE: .C1, .C2, etc.=Chapter 1, Chapter 2, etc.

**Over 30 TEXT files included is ASM (assembler)-PASCAL-PIC (position independent code) TSC BASIC-C, etc.

Foreign Orders Add $4.50 S/H

Softcover — Large Format

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

Continually Updated In 68 Micro Journal Monthly

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615) 842-4601
Telex 5106006630

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware and Motorola
"68' Micro Journal is a trademark of Computer Publishing Inc.

MUSTANG-020 Super SBC™

***

We Proudly Announce the MUSTANG-020 Super SBC*
"The one with the **REAL KICK!**"
Only from DATA-COMP

This is the NCC, world beater GMX SBC, in a super configuration. Data-Comp has mated it to a power plus power supply/stylish cabinet and your choice of floppy and/or hard disk drives. Available in several different configurations. (1) single board. (2) single board and regulators for your cabinet or mainframe and power supply. (3) single board - power supply and cabinet - your disk drives. (4) single board - power supply/cabinet - our drives configured to your specs, and ready to run. OS9 68K will be available as well as several other popular operating systems. Also all the popular OS9 68K software and Motorola 020-BUG will be available at a very reasonable price.

Note: We will include the Motorola 020-BUG at no additional charge. This item alone sells for $500.00. This allows direct coding of 68020 advanced codes. 020-BUG is required for our version of OS9. Please bear in mind, this system is the one others are compared to.

This system is the state-of-the-art star-ship. It runs rings around any other 68XXX SBC, and most mainframes. The speed and expanded RAM make this the "best buy" by a far stretch! A true multi-user, multi-tasking computer. So far advanced that even the experts don't call it a micro. Compared to the others, it isn't even a "horse race." And the price is certainly right. You can bet on this one!

So, will it be **Turtle** or **Thoroughbred?**

MUSTANG-020 System Prices
Effective November 1985

Mustang-020 SBC, wired & tested with 4 DB25 Serial ports pre-wired, ready to install with your cabinet, P/S, CRT and drives.....$2750.00

M020 Cabinet and P/S, for Mustang-020, less cables..... $269.95

M020 Cables, dual floppy or winchester, specify which - floppy or winchester.....$39.95

M020FC Floppy cabinet and P/S, holds and powers 2 thin-line floppies.....$79.95

M020F Floppy, 80 track, DD/DS.....$269.95

OS-9, SPECIAL Mustang-020 version.....$350.00

MC68881F/P co-processor.....$495.00

10 Megabyte Winchester.....$695.00

20 Megabyte Winchester.....$895.00

Winchester Controller (Xebec-).....$395.00

Note: for orders of complete systems (Mustang-020, cabinet & P/S, disk drives and OS-9, deduct 5% from total package. (limited time offer) See opposite page.

** Special Winchester Notice **

The Mustang-020 device descriptors will allow you to use practically ANY winchester drive supported by XEBEC controllers.

* * *

Include: $3.50 SBC, cables only S/H. Cabinets include $7.50 S/H. Complete System include $20.00. All checks must be in USA funds. Overseas specify shipping instructions and sufficient funds.

Dealer & Quantity Discounts Available

* Mustang-020 is trademark of Data-Comp-CPI

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
TELEX 5106006630

MUSTANG-020 Super SBC™

Mustang 020 Features

- 12.5 MHz MC68020 full 32-bit wide path Processor
  - 32-bit wide non-multiplexed data & address buses
  - On-chip instruction cache
  - Object-code compatible with earlier M68000 family processors (68000/68008/68010)
  - Enhanced instruction set - Coprocessor interface
- Optional 68881 Floating point Coprocessor (12.5 MHz)
  - Direct extension of 68020 instruction set
  - Full support of IEEE P754, draft 10.0
  - Transcendentals and other math functions
- 2 Megabytes of RAM (512K x 32-bit organization)
- Up to 256K bytes of EPROM (64K X 32-bits)
  - Uses four 2764, 27128, 27256, or 27512 EPROMs
- 4 Asynchronous serial I/O ports (2 x MC68681 DUART)
  - Software programmable baud rates to 19.2K
  - Standard RS-232 interface
  - Optional network interface on one port
- Buffered 8-bit Parallel I/O Port (1/2 MC68230)
  - Centronics-type parallel printer pinout
  - May also be used as parallel input port
- Expansion Connector for Additional I/O Devices
  - 16-bit data path
  - 256 byte address space
  - 2 interrupt inputs
  - Clock and Control Signals
- Time-of-Day Clock/Calendar w/battery backup
- Controller for up to Two 5 1/4" Floppy Disk Drives
  - Single or double sided
  - Single or double density
  - 48 or 96 tracks per inch (40/80 Track)
- Mounts Directly to a Standard 5 1/4" Disk Drive
- SASI interface for Intelligent Hard Disk Controllers
- Programmable Periodic Interrupt Generator
  - For time-slicing and real-time applications
  - Interrupt rates from microseconds to seconds
  - Highly Accurate timebase (5 PPM)
- 5-bit sense switch, readable by the processor
- Hardware single-step capability

*** And many other advanced features not found on other systems priced much higher.

MUSTANG-020 Benchmarks **
Time Seconds

Type System	32 bit Tot. Loop	Register Long Loop
IBM AT 7300 Xenix Sys 3	9.7	No Registers
AT&T 7300 UNIX PC 68010	7.2	4.3
DEC VAX 11/780 UNIX Berkley 4.2	3.6	3.2
DEC VAX 11/750 " " "	5.1	3.2
68000 OS9 68K 8 Mhz	18.0	9.0
68000 " " 10 Mhz	6.5	4.0
MUSTANG-020 68020 MC68881 OS9 16 Mhz	2.2	0.88
MUSTANG-020 68020 MC68881 UNIFLEX "	1.8	1.22

** Loop:
```
Main()
{
  register long i;
  for (i=0; i < 999999; ++i);
}
```

Estimated MIPS - MUSTANG-020 - 2.5 MIPS
Motorola Specs: Burst up to 7 - 8 MIPS - 16 Mhz

Note: The standard MUSTANG-020 is now available with the TSC UniFLEX System. TSC software is optimized for the 68020 and MC68881 Co-Processor. This includes the TSC "C" compiler and other software. Speeds are greatly enhanced!

Mustang-020 Software

OS-9

OS-9	$350.00
BasicO9	3 0.0
C Compiler	400.00
Fortran 77	*4 0.00
Pascal Compiler	4 0.00
OMEGASOFT-PASCAL	900.00
SCULPTOR+	*Call
CUM	*Call

UniFLEX

UniFLEX	$450.00
Screen Editor	150.00
Sort-Merge	200.00
BASIC/PreCompiler	300.00
C Compiler	350.00
COBOL	750.00
Fortran 77	450.00
SCULPTOR+	*Call

* New items

prices/delivery dates subject to change.

SCULPTOR+..We are USA distributors for SCULPTOR+. Call or write for site license or multiple discounts.

Call for additional information. New Software being added monthly!

Standard Systems
MUSTANG-020 Prices (Nov 13,1985) Net

	Dual 5" 80 trk. Floppy No Winchester		Winchester & 1 Floppy
020 Board	$2750.00		$2750.00
Cabinet	269.95		269.95
5"-80 trk Floppy(2)	539.90	(1)	269.95
Floppy cable	39.95		39.95
OS-9 68K	350.00		350.00
		Winchester cable	39.95
Total System	$3949.80	Winchester controller	395.00
Less 5%	-197.49	10 MegByte Winchester	695.00
	$3752.31	Total System	$4809.80
S/H UPS	20.00	Less 5%	-240.49
			$4569.31
Total	$3772.31	S/H UPS	20.00
		Total	$4589.31
		With 20 MegByte Winchester Add:	200.00
			$4789.31

NOTE: 68881 Co-Processor Add $495.00
UniFLEX $450.00
less $350.00 (OS-9) Add $100.00

Prices and Specifications subject to change.

FLEX User Notes

Ronald W. Anderson
3540 Sturbridge Court
Ann Arbor, Mi 48105

BUGS, BUGS, BUGS

What a month (or two). I've been involved, no, surrounded by programs with bugs to find and eliminate. I've just finished PAT, the editor and my first test users at first reported a number of bugs that have been stamped out. A few more reports are coming in and I still have a few bugs that have been reported by only one user. Whether the problem is one of understanding of the manual or with the software in a particular hardware system configuration environment, it makes no difference. The problems must be tracked down and eliminated. I must admit to being happy that those bugs have trailed off into a now-and-then status, because just about the time that happened, a large program on a machine at work was needed, and two of us have been busily debugging for a solid two weeks now. Of course the writing of the program (actually two, since the system uses two 6809 processors) was the first hurdle. One processor has around 22K of PL/9 code and the other about 14K. Finally today, we reached the point of the "fun". That is, where most everything works fine, but there is still some "clean-up" to do.

As if that were not enough to keep me busy, we have a new exchange student (high school) who is having trouble with Algebra, mostly because of the difficulty in understanding English, though he speaks English quite well. Fortunately, Algebra is one of my stronger subjects, and I have been able to help, illustrate, do step-by-step examples, and explain 6 different ways until finally there is understanding.

About the time the big project hit at work, I finally had some hardware in my hands to complete a consulting project that had been stalled all summer. I am also in the middle of that debug, though at this point, I am also nearly done with that. I have about 30 lines of particularly knotty code to finish, and a little bit of analog circuit design fine tuning to accomplish, and that project will be done.

I was just adding up the code (all, incidentally in PL/9). PAT is about 16.5K at the moment. The work project, as I said, 22K, and the consulting job about 7K. In terms of source listing, I suppose the three total 90 pages or so (excluding the library routines that are pretty well standard). Thanks, Windrush and Graham Trott, for PL/9. Had I to do all these projects in Assembler in the short time that they have been done, I would probably have finished only the smallest of them by now.

Reader Responses

I continue to get some further responses from readers, though answers to my appeal of last June have tapered off to a small dribble. One request that interests me a great deal is for a discussion of "good programming practice". I don't suppose there are two "experts" who would agree completely on the subject, but certainly nearly all agree on one point, that of writing programs modularly.

One nice feature of the more "modern" languages is the "local variable". Pascal, and "C" in the standard language category, and PL/9 and Whimsical in our little FLEX environment, are all languages that support this concept. I won't get into a complete discussion of the various "storage classes" in "C". (Bud Pass, take note that such a discussion would make an interesting topic for a future column). Let me just explain the usefulness of, and the distinction between, Local and Global variables.

Standard BASIC permits only Global variables. That is, once a variable is used for something, it is available to any part of the program. Suppose for a moment that I have used the variable RA in one part of a BASIC program to hold the RAdius of a circle or arc. Later I have a variable that holds the RAtio of two numbers and I forget that I have already used RA elsewhere, so I use RA here again. BASIC doesn't complain at all, but at the end of the program the variable contains the value of RAtio and the value of RAdius has been lost. If I print RA expecting to see the value of RAdius and I get the value of RAtio, the results will be nonsense.

The "structured" languages allow you to define variables as GLOBAL, or existing everywhere in a program, or as LOCAL, existing in only one section of a program (read "section" as PROCEDURE in Pascal or function in "C".) The best way to keep out of trouble in writing a large program is to use as many LOCAL variables as possible and as few GLOBAL variables as possible. Suppose I am writing a PL/9 program and I "declare" a GLOBAL variable:

```
GLOBAL
  BYTE CHAR;
```

Now if I use CHAR to hold the value of a CHARacter in two different procedures of the program, I might have some surprising results. That is, one procedure stores a value in CHAR, and another routine changes that value. It is the same as the case above of using one variable for two different purposes in a BASIC program. If I declare the character as a LOCAL variable or as a "formal parameter", it is used only in the procedure in which it is declared. It is "created" when that procedure is called, and destroyed when program control returns from that procedure. An example is worth a lot of words. Suppose I am writing an input and output character procedure to be compatible with FLEX.

```
PROCEDURE PUTCHAR(BYTE CHAR);
   ACCA = CHAR;
   CALL $CD18;
ENDPROC;


PROCEDURE GETCHAR: BYTE CHAR;
   CALL $CD15;
   CHAR = ACCA;
ENDPROC BYTE CHAR;
```

Another indispensible feature of modern compilers is that of "passing parameters" to sub procedures (functions or procedures) and the ability to return values from such procedures or functions. Suppose I have a general procedure to calculate the volume of a cylinder. I need

only the RADIUS and the HEIGHT as input, and I will get VOLUME back. I would define the procedure as:

```
PROCEDURE VOLCYL(REAL RADIUS, HEIGHT): REAL VOLUME;
   VOLUME = PI * RADIUS * RADIUS * HEIGHT;
ENDPROC REAL VOLUME;
```

The above example presupposes that PI has been defined as a real number 3.14159265..... I would use the procedure in a program by using the following statement:

```
VOLUME = VOLCYL(RAD, HT);
```

Note that I don't have to use variables named RADIUS and HEIGHT in the line that uses the procedure I could just as well have said:

```
VOLUME = VOLCYL(1.5,6.75);
```

though it would be rather pointless to use the procedure to calculate the volume of a cylinder whose dimensions are known and fixed at the time the program is written.

You might notice the similarity between these "procedure or function" calls and the FUNCTION definition in BASIC. In fact, a function call in Pascal, "C", PL/9 or Whimsical looks exactly like a call to one of the "intrinsic" functions in BASIC. If we define a function SIN(REAL X); in PL/9 to return the value calculated, the call in the program will look exactly like the call of the intrinsic function SIN in BASIC. Y = SIN(X);. The only difference is the semi-colon that terminates the statement in Pascal, "C", or PL/9.

The items passed to the procedures are called Parameters. RAD, HT in one example and 1.5,6.75 in the other, are parameters. Note that if a procedure or function returns a value, you can't simply say VOLCYL(X,Y). You must treat the call as returning a value which must be asigned by an assignment statement as in Z=VOLCYL(X,Y);

Returning to our example of input and output character procedures, I output a character by the statement PUTCHAR(ch); Where "ch" stands for the character I want to output. I can use 7, which will ring the "bell" on the terminal, 'B, which will print a "B" on the terminal, a symbolic constant defined previously such as CR or LF, or the value of a BYTE variable such as CH. The variable CHAR exists only within the procedure PUTCHAR (that is the variable named CHAR that is the "formal parameter" for that procedure.) When I use the statement PUTCHAR(ch), the value of "ch" is "pushed" on the machine stack and the program jumps to the "subroutine" PUTCHAR, which uses the value "passed to it" on the stack and outputs that character.

I input a character by the statement VAR = GETCHAR;. VAR can be any byte variable whose value I want to be set equal to a character input from the terminal. The PROCECURE GETCHAR has a LOCAL variable declared by the part of the PROCEDURE definition ": BYTE CHAR;". When the main program jumps to the subroutine GETCHAR, there is space made on the stack for the local variable, which is used to hold the value of the contents of the A Accumulator after calling the FLEX routine to get a character from the terminal in the A accumulator. The last line of the Procedure indicates that it is to return the value of CHAR to the calling program. In PL/9, a byte value is returned in ACCB. The compiler handles the code to allocate and deallocate the space on the stack for parameters passed to procedures and values returned.

The point of all this is that I can use the identifier "CHAR" in two entirely different procedures, and the uses can not interfere with each other. Whatever GETCHAR puts into its CHAR variable can't interfere with what PUTCHAR has passed to it in its CHAR veriable. GETCHAR and PUTCHAR cannot possibly interfere with each other.

I know this explanation has been sketchy and perhaps used some terms you are not familiar with, but perhaps you can see the possibilities. You use LOCAL variables wherever possible, and break your programs into small "chunks" called PROCEDURES. If you are careful to pass your procedures the information they need as input, and to return the results, your procedures can be "independent" of each other so that an error in one can't possibly cause a strange symptom when you execute some remote part of the program. If you use procedures that are no larger than a page or so of code, they are not difficult to comprehend when you read them, and if you use parameters properly, you can test procedures individually. In fact, when I write a program, I generally start writing the procedures I know that I will need, and I test each one with a "main program" that eventually is replaced with my "real" main program after all the procedures are tested.

At work, I can use procedures from previously written programs, and know that they will work together regardless of variable name conflicts within the local variables or parameter names. Some procedures become so standard that they are made part of a "library" package that can be included in our programs. The use of local variables in all procedures that are so standard as to become part of a library, makes it possible to put a program together with several different library files, with no chance of variable name conflicts or problems.

One point that might not be obvious in all the above discussion, is that veriables that exist "temporarily" while a particular procedure is being executed, don't take up variable storage space on a permanent basis. Only GLOBAL variables have space reserved during the entire execution of the program. This reduces the amount of RAM required while running the program. The main reason to use local variables, and to pass parameters to proceduree, however, is not to save RAM space, but to make use of well defined modules with well defined connections to the other parts of the program. I claim that a 5 page program in BASIC is about 25 times as hard to debug as a 1 page program, while a 5 page program in a structured language, consisting of 5 well defined procedures of a page each, is only 5 times as hard to debug as a 1 page program. Each procedure can, in fact, be debugged as a separate 1 page program.

One point that is not obvious, is that in most of the structured languages, there may be a Global variable of the same name as a Local one. In all cases, the local one takes precedence. That is if there is a Global variable COUNT and a local variable COUNT in a particular procedure, a reference to COUNT within that procedure will be to the local variable. If another procedure does not have COUNT defined locally, a reference to COUNT in it will access the Global variable COUNT. This is a very reasonable way to handle this situation, though now and then having a local and a global variable of the same name can lead to confusion and mysterious bugs.

How about some examples of simple procedures?

```
PROCEDURE ABS(REAL NUMBER);
  IF NUMBER <0 THEN NUMBER =-NUMBER;
ENDPROC REAL NUMBER;

PROCEDURE COS(REAL ANGLE);
ENDPROC REAL(SIN PI/2-ANGLE);

PROCEDURE RAD_TO_DEC(REAL RADIANS) : REAL DEGREES;
   DEGREES = RADIANS * 180/PI;
ENDPROC REAL DEGREES;
```

You can "call" a procedure with a variable name, a constant, or an expression as the parameter. For example in the case of RAD_TO_DEG above, you could say:

```
ANGLE = RAD_TO_DEG(ATAN(Y/X));
```

That statement would first divide Y by X, then find the angle whose TANGENT is equal to the result of the division (in radians) and then convert the angle in radians to degrees and assign it to the variable ANGLE.

Disk Drive Troubles

I have a pair of Tendon 40 track double sided drives, and one had suddenly gone crazy. It would suddenly not be able to read a disk at all. I would push the head around a little, testing for binding or the like, and it would suddenly work again. It finally went permanantly bad, but one day I accidentally formatted a disk on it and found the disk to have formatted just fine. I copied some files to it and then put the disk in another drive which couldn't read it at all. BINGO! The head was getting out of alignment. After suspecting that the "taut band" had gotten stretched somehow, I finally traced it down to a loose setscrew on the taut band drum where it is clamped on the stepper motor shaft. I tightened the setscrew and realigned the head and the drive has been working fine ever since.

You read right, I realigned the head. How? First of all insert a disk made on a known good drive (maybe there is a fallacy there, better to use an alignment disk). On the Tandon, there is a whole section of the drive base casting that is separate from the rest. It is the part at the rear center that holds the head stepper and head support rails. Loosen three allen head cap screws, one on the top rear, and two on the bottom near the large drive pulley, and the head alignment is adjustable via an offset cam on the top rear. The cam has a screw slot and looks like it could be a screw head. Rotating it causes it to force the smaller head stepper casting to move with respect to the larger one. Rotate it slightly in each direction until you find that you can read the disk. Try reading a long text file near the middle track. If you can't read it for a long time, Flex will give up re-trying and report an error. Just try to LIST it again and continue. You will find a setting at which the disk can be read (if head alignment is the problem with the drive).

Now find the limits of adjustment where the drive fails to read the file. Approach the adjustment in the same direction of rotation of the adjusting screw at all times. For example, you may find that the drive starts to read the file with the screw slot at 9:00 and stops reading it with the slot pointing at 10:30, approaching both settings by turning the screw clockwise. Now just back the screw around CCW and turn it clockwise carefully to 9:45 by your best estimate. Better, make two marks on the casting with a pencil at the two limits of operation, and set the adjustment midway between these two limits. Now lock the casting down with the three cap screws and check the drive. Format a disk in it and copy a full disk to it from a good drive. Now put the newly made disk in the good drive and see if you can read all tracks. You might want to do the test with a "system disk" so you can see if you can boot with the disk. If you can read all tracks of a disk from a known good drive in the "repaired" one, and you can read all tracks of a disk written in the repaired one on a known good drive, you have succeeded.

Other drives adjust head alignment by loosening the screws that hold the stepper motor to the base casting. In those units, the screws that hold the steppr fasten it through slots in tabs on the stepper. You align the head by roteting the stepper slightly. You can again mark a reference point on the end of one of the ears, and mark the limits at which it reads a track, and split the difference. My philosophy is that if it doesn't work anyway, I am not going to make it much worse by trying to fix it. If I am not successful, I can always bring it to a repair facility.

Though the problem with my drive turned out to be a loose screw, I at first suspected that the taut band had slipped on the rivet that fastens it to the head assembly at one end. The band was a little puckered around the rivet, and I put some 5 minute epoxy around the rivet head and under the band at the rivet.

I note that FLEX 2.8:3 goes through a sequence when it has disk read errors. It is pretty smart in that when you put a single density or single sided disk in the drive (or a 40 track disk in an 80 track drive) FLEX finds the right combination of sides, tracks, and density, and can read the disk after a few tries. If you have 40 track drives, you may note that FLEX tries to step 80 tracks as one of the tests. That means that the head is stepped to its travel limit, and then beat against that limit for another 40 steps. (The stepper motor stalls, but the head assembly is driven into the stop with some significant force for each step). I think this aggravates the situation of anything being loose in the head positioning components of the drive. Of course if you have 80 track drives, it doesn't happen. If you do take your drive to a repair shop, try to find one where there is some reason to believe that the person who is going to do the work is competent, or your drive will end up going back to the manufacturer, possibly for many months.

Time and Date

A reader wrote and asked for a column devoted to how to interface a clock calendar board to FLEX so that FLEX can read the date from the calendar chip rather than ask the user for it on boot. I have two systems running in that mode with two different FLEX versions, and I can say from experience that it is no small trick to accomplish the job, particularly since no two clock chips use the same format or method of reading the date, and no two versions of FLEX have the code that must be patched in the same place. What I can and will do, is publish my code for a particular chip and FLEX 2.7:3. The code will work with fairly simple modifications with another FLEX, and the chip specific code will be a good pattern for another chip.

If you want to get a head start, let me give you some code to look for. First boot FLEX and then hit your system RESET button. Now use SBUG-E EXAMINE command E to look at memory. FLEX COLDStart is a jump at $CD00, so first E CD00-CDFF. The code at CD00 will be 7E NN NN where the four hex digits NNNN represent an address where the warmstart code begins. Now jot that address down and go through FLEX looking for the text string DATE MM/DD/YY. Note the address of the start of the string. Somewhere in the COLDStart code you will find the code to load X with the address of that string and JSR or LBSR PSTRNG. PSTRNG is at $CD1E. If you can locate that point, half the battle is over. If I have lost you already, better find someone else to do the job for you. A simple disassembler and / or the Motorola 6809 "Reference Card" will be of much help in deciphering the instructions in FLEX. When you trace the COLDStart code, follow the program flow from the JUMP instruction at $CD00. If you come to a JUMP or BRA or LBRA instruction, follow to its destination and continue. If you come to a BSR or LBSR, go and trace through the subroutine and continue after the BSR instruction after going through the subroutine. The FLEX Advanced Programmer's Guide will be of much help in finding out what some of the subroutines are.

Well, like the old Saturday afternoon Serial Movie, I am going to leave you hanging this time. Next time we will look at the patches to FLEX and how to implement them. TSC says FLEX is not Patchable. They mean that you can't append a patch to the FLEX.SYS file and have it overlay areas of FLEX. You can, however, patch FLEX by more devious means such as using the PIX utility to change bytes on the disk directly.

"C" User Notes

Edgar M. (Bud) Pass, Ph.D.
1454 Latta Lane
Conyers, Ga 30207

Readers have been providing C programs for publication and I have collected several useful C programs and subroutines, but the space for publication has been limited. This chapter presents several of these programs and functions, which hopefully will be useful and will illustrate the use of the C language. Whenever the local C bulletin board is up, I will put the string library and other programs on it.

DATE PACKAGE

Conversion of date formats has always been complex, partially due to the historical interference with the calendar and partially due to the complicated relationships between the seasons and the solar and lunar days. The functions provided below assist in the conversion of date formats, and in such determinations as the number of days between two dates, the date a number of days after another date, the day of the week of a date, etc.

The three most-commonly used date formats are the following:

Gregorian	mm/dd/yyyy or yyyy/mm/dd
Julian	yyyyddd
Astronomical	nnnnnnn

For those not familiar with astronomical date format, it is simply a sequential numbering of days, starting with some point in the distant past. It is especially useful for determining the number of days between dates, the date a given number of days before or after another, etc.

If a year less than 100 is input to any of the functions, it is assumed to be within the current century. If an error is detected in an input parameter, zero is returned.

```
#include <stdio.h>

main(argc,argv)
int argc;
char *argv[];
{
 long calcsday();

 switch (atoi(argv[1]))
 {
 case 1:
  printf("%d\t%d\t%d\t%d\t%d\n",1,
   calcwday(atoi(argv[2]),atoi(argv[3]),atoi(argv[4])),
   atoi(argv[2]),atoi(argv[3]),atoi(argv[4]));
  exit(0);
 case 2:
  printf("%d\t%d\t%d\t%d\t%d\n",2,
   calcyday(atoi(argv[2]),atoi(argv[3]),atoi(argv[4])),
   atoi(argv[2]),atoi(argv[3]),atoi(argv[4]));
  exit(0);
 case 3:
  printf("%d\t%d\t%d\t%d\t%d\n",3,
   calcmoday(atoi(argv[2]),atoi(argv[3])),
   atoi(argv[2]),atoi(argv[3]),0);
  exit(0);
 case 4:
  printf("%d\t%d\t%d\t%d\t%d\n",4,
   calcymday(atoi(argv[2])),
   atoi(argv[2]),atoi(argv[3]),0,0);
  exit(0);
 case 5:
  printf("%ld\t%d\t%d\t%d\t%d\n",5,
   calcsday(atoi(argv[2]),atoi(argv[3]),atoi(argv[4])),
   atoi(argv[2]),atoi(argv[3]),atoi(argv[4]));
  exit(0);
 }
 pflinit(); /* for mc cosh c only */
}

/*
**   date subroutines
**
**   see cacm oct 68 p 657, oct 70 p 621, oct 72 p 918.
*/

/*
**   this function calculates the day of the week
**   from the year, month, and day of month.
*/
int calcwday(year,month,day)
int year,month,day;
{
    long t1,t2,t3,t4,t5,t6,t7,t8;

    if ((year < 0) || (month < 1) || (month > 12) ||
        (day < 1) || (day > 31))
        return 0;
    if (year < 100)
        year += 1900;
    t1 = month + 10;
    t2 = t1 / 13;
    t3 = (13 * (t1 - t2 * 12) - 1) / 5 + day + 77;
    t4 = year - (14 - month) / 12;
    t5 = t4 / 100;
    t6 = t4 / 400;
    t7 = (5 * (t4 - t5 * 100)) / 4;
    t8 = t3 + t7 + t6 - t5 - t5;
    return (t8 % 7 + 1);
}

/*
**   this function calculates the day of the year
**   from the year, month, and day of month.
*/
int calcyday(year,month,day)
int year,month,day;
{
    long t1,t2,t3,t4,t5,t6;

    if ((year < 0) || (month < 1) || (month > 12) ||
        (day < 1) || (day > 31))
        return 0;
    if (year < 100)
        year += 1900;
    t1 = (3055 * (month + 2)) / 100;
    t2 = 2 * ((month + 10) / 13) + 91;
    t3 = (year % 4 + 3) / 4;
    t4 = (year % 100 + 99) / 100;
    t5 = (year % 400 + 399) / 400;
    t6 = ((1 - t3 + t4 - t5) * (month + 10)) / 13;
    return (t1 - t2 + t6 + day);
}
```

```
/*
**   this function calculates the month and day of the month
**   from the year and day of year; the month is shifted
**   left five bits and added to the day of the month so that
**   both may be returned simultaneously.
*/
int calcmodny(year,yday)
int year,yday;
{
    long t1,t2,t3,t4,t5,t6;

    if ((year < 0) || (yday < 1) || (yday > 366))
        return 0;
    if (year < 100)
        year += 1900;
    t3 = (year % 4 + 3) / 4;
    t4 = (year % 100 + 99) / 100;
    t5 = (year % 400 + 399) / 400;
    t2 = (1 - t3 + t4 - t5);
    t3 = ((yday > (59 + t2)) ? (yday + 2 - t2) : yday);
    t1 = ((t3 + 91) * 100) / 3055;
    t6 = t3 + 91 - (t1 * 3055) / 100;
    return (((t1 - 2) << 5) | (t6 & 31));
}

/*
**   this function calculates the year/month/day from the
**   astronomical day number; the year is shifted left
**   nine bits and the month is shifted left five bits so
**   that all three may be returned simultaneously.
*/
long calcymday(ast)
long ast;
{
    long t1,t2,t3,t4,t5;

    t4 = ast + 68569;
    t5 = (4 * t4) / 146097;
    t1 = (146097 * t5 + 3) / 4;
    t4 = t4 - t1;
    t1 = (4000 * (t4 + 1)) / 1461001;
    t2 = (1461 * t1) / 4;
    t4 = t4 - t2 + 31;
    t2 = (80 * t4) / 2447;
    t3 = (2447 * t2) / 80;
    t3 = t4 - t3;
    t4 = t2 / 11;
    t2 = t2 + 2 - 12 * t4;
    t1 = 100 * (t5 - 49) + t1 + t4;
    return ((t1 << 9) | (t2 << 5) | (t3 & 31));
}

/*
**   this function calculates astronomical day number
**   from year, month, and day of month.
*/
long calcaday(year,month,day)
int year,month,day;
{
    long t1,t2,t3;

    if ((year < 0) || (month < 1) || (month > 12) ||
        (day < 1) || (day > 31))
        return 0;
    if (year < 100)
        year += 1900;
    t1 = (month - 14) / 12;
    t2 = (1461 * (year + 4800 + t1)) / 4;
    t3 = (367 * (month - 2 - t1 * 12)) / 12;
    t1 = (3 * ((year + 4900 + t1) / 100)) / 4;
    return (day - 32075 + t2 + t3 - t1);
}
```

SET FUNCTIONS

Jeff Craig has provided several functions which perform set operations on discrete sets with a finite number of members. They perform the following set operations:

- intersection of two sets
- difference of two sets
- union of two sets
- duplicate member elimination
- ordering of members

Set elements are represented by characters in strings. As is the custom in C, sets are null-terminated. It is the responsibility of the calling functions to assure that the strings are long enough to contain the resulting sets.

Jeff also sent a program he calls "sherlock" which performs a limited amount of inferential logic using set operations. It is too lengthy to print here, but it will be available on the bulletin board.

```
#include STDIO.H
#include EXTIO.C
#include CTYPE.H
#include SETS.C
#define SIZ 20

main()
{
     char setx[SIZ];
     char sety[SIZ];
     char setz[40];
     printf("\n");
     printf("get set A\n");
     stuf(setx);
     printf("\n");
     printf("get set B\n");
     stuf(sety);
     printf("\n");
     displ(setx);
     printf("\n");
     displ(sety);
     printf("\n");
     union(setx,sety,setz);
     printf("the union\n");
     displ(setz);
     printf("\n");
     isort(setz);
     printf("sorted\n");
     displ(setz);
     printf("\n");
     nodup(setz);
     printf("no duplicates\n");
     displ(setz);
     differ(setx,sety,eetz);
     printf("\n");
     printf("difference\n");
     displ(setz);
     intera(setx,sety,setz);
     printf("\n");
     printf("intersection\n");
     displ(setz);
     printf("\n");
}

stuf(seta)
char seta[];
{
     int i, c;
     i = 0;
     c = 0;
     while(c != '\n')[
          c = getchar();
          seta[i] = c;
          ++i;
     ]
     --i;
     seta[i] = '\0';
}
displ(seta)
char seta[];
{
     printf("%s",seta);
}

/* intersection of two sets */
intera(seta,setb,setc)
char seta[];
char setb[];
char setc[];
{
     int i, x, w;
     i = 0;
     x = 0;
```

```
     w = 0;
     while(aetb[x] != '\0'){
     for(i = 0; aeta[i] != '\0'; ++i){
          if (aeta[i] == aetb[x]){
               aetc[w] = aeta[i];
               ++w;
          }
     }
     ++x;
     }
     aetc[w] = '\0';
}

/* difference between two aeta */
differ(aeta,aetb,eetc)
char aeta[];
char aetb[];
char eetc[];
{
     int i, x, w, fleg;
     i = 0;
     w = 0;
     while(aeta[i] != '\0'){
     fleg = 0;
     for(x = 0; eetb[x] != '\0'; ++x){
          if (eeta[i] == aetb[x]){
               flag = 1;
               break;
          }
     }
     if (flag == 0){
          eetc[w] = eata[i];
          ++w;
     }
     ++i;
     }
     aetc[w] = '\0';
}

/* union of two aeta */
union(aata,aetb,aetc)
char aeta[];
char aetb[];
char aetc[];
{
     int x, i;
     for(x = 0; aeta[x] != '\0'; ++x){
          aetc[x] = aeta[x];
     }
     for(i = 0; aetb[i] != '\0'; ++i){
          aetc[x] = aetb[i];
          ++x;
     }
     aetc[x] = '\0';
}

/* eliminate duplicate members within a set */
nodup(aeta)
char aeta[];
{
     int x, y, i;
     x = 0;
     while(aeta[x] != '\0'){
     for(i = x + 1; aeta[i] != '\0'; ++i){
          if (aeta[i] == aeta[x]){
               y = i;
               while(aeta[y] != '\0'){
                    aeta[y] = aeta[y + 1];
                    ++y;
               }
               --i;
          }
     }
     ++x;
     }
}

/* aort */
iaort(aeta)
char aeta[];
{
     int i, j, t, uplim;
     i = 0;
     while(aeta[i] != '\0'){
          ++i;
     }
     uplim = i;
     for(i = 1; i < uplim; ++i){
          j = i;
          t = aeta[j];
          while(j > 0 && aeta[j-1] > t){
               aeta[j] = aeta[j-1];
               j = j - 1;
          }
          aeta[j] = t;
     }
     aeta[i] = '\0';
}
```

PERMUTATION GENERATOR

This C program generates all permutations of the characters of strings input to it. Since the number of permutations of a string of length n is n! (1 x 2 x 3 x ... x n), the program limits the length of the input string to 10; however, the algorithm could produce the permutation of any length string, if the work areas are dimensioned appropriately. This number grows very quickly, making the output of longer string permutation sets quite time-consuming. Assuming that the string is printed with two delimiters (spaces or CR-LF), the number of characters required to generate all permutations of a string of length n is (n! x (n + 2)). The following table provides the approximate times to output the permutations of several string lengths, assuming a 9600 Baud terminal:

length	characters	seconds
2	8	.008
3	30	.031
4	144	.150
5	840	.875
6	5760	6.000
7	45360	47.250
8	403200	420.000
9	3991680	4158.000
10	43545600	45360.000

One simple application of this program is to help solve the "Jumble" puzzle in the paper. Since the strings in that puzzle are of lengths 5 and 6, the direct generation of all permutations can be very effective.

```
#include <stdio.h>

main()
{
    short int p[11], d[11], c, f, n, m, i, k, q;
    char x[11], t;

    while (1)
    {
        printf("\nEnter string to be permuted. ");
        if ((fgets(x,f = 10,stdin) == NULL) ||
            (!(m = strlen(x) - 1)))
            exit(0);
        for (c = i = x[m] = 0; i < 11; ++i)
        {
            p[i] = 0;
            d[i] = ((i <= m) ? i : 0);
        }
        do
        {
            k = 0;
            n = m;
            do
            {
                if ((q = p[n] += d[n]) == n)
                    d[n] = -1;
                else
                {
```

```
                if (q)
                {
                    n = -3;
                    break;
                }
                d[n] = 1;
                ++k;
            }
        }
        while (--n >= 2);
        if (n > -2)
        {
            q = 1;
            f = 0;
        }
        t = x[q += k];
        x[q] = x[q - 1];
        x[q - 1] = t;
        if ((c += 1 + m) >= 80)
        {
            c = m + 1;
            printf("\n");
        }
        printf("%m ",x);
    }
    while (f);
    printf("\n");
  }
}
```

C PROBLEM

The previous C problem was to write a program which will generate a fixed, variable-length, response (based upon the contents of a file) to a keyword or phrase input to it. The simplest manner in which to solve the problem would be to read the entire file into memory. However, this solution would severely limit the flexibility of the program. Instead, the program keeps only the keywords or phrases in memory and uses the random file access functions described in the previous chapter to position the input file appropriately.

```
/*
** help program      called as:     help helpfilename
**   help file has following format:
**      *keyword
**      text
**      :
**      text
**      *keyword
**      text
**      :
**      text
**      *      (FLEX users must end with dummy keyword)
*/

#include <stdio.h>

#define TABSIZE 256
#define TABENT  64

main(argc,argv)
int argc,**argv;
{
 FILE *input;
 int i = 0, j, k;
 char string[128], query[128], keyword[TABENT + 1][TABSIZE + 1];
 long loc[TABSIZE + 1];

 if (argc < 2)
 {
  fputs("usage: help filename\n",stdout);
  exit(1);
 }
 if ((input = fopen(*++argv,"r")) == NULL)
 {
  fputs("Can't open ",stdout);
  fputs(*argv,stdout);
  fputs(".\n",stdout);
  exit(2);
 }
 setbuf(input,NULL);
 while (fgets(string,128,input) != NULL)
  if ((*string == '*') && (*(string + 1) >= ' '))
   if (i >= TABSIZE)
    break;
   else
   {
    string[strlen(string) - 1] = 0x00;
    strncpy(keyword[i],string + 1,TABENT);
    loc[i++] = ftell(input);
   }
 rewind(input);

 while (1)
 {
  fputs("Enter keyword. ",stdout);
  if ((fgets(query,128,stdin) == NULL) ||
   ((j = strlen(query) - 1) < 1))
   break;
  query[j] = 0x00;
  for (j = 0; j < i; ++j)
   if (!strcmp(query,keyword[j]))
   {
    if (!fseek(input,loc[j],0))
    {
     while ((fgets(string,128,input) != NULL)
      && (*string != '*'))
      fputs(string,stdout);
    }
    break;
   }
 }
 fclose(input);
 exit(0);
}
```

For the next C problem, modify the program just presented to ignore character case, more than one internal space in a keyword or phrase, and leading and trailing spaces in a keyword or phrase. Also add wildcards to the match criteria such that an "*" in the request keyword matches any number of characters and "?" matches any single character, then output all matching keywords and associated text in the file.

EXAMPLE C PROGRAM

Following is this month's example C program; it plays a board game.

```
/*      brain.c -- brain teaser.
 *
 *      from "Computers in Mathematics: a Sourcebook of Ideas",
 *      edited by David H. Ahl.
 *      "Brain Teaser", Hal Knippenberg, page 48.

The object of this puzzle is to change the patterns
of 0's and 1's until the board has a 0 in the center
and 1's in all other positions.
        To change the board pattern, enter the
number of a square that contains a 1.  Enter the
squares position number as follows:

        1  2  3
        4  5  6
        7  8  9

        Choosing a square in the center of an edge
(2,4,6,8) causes all positions along the edge to
change state.  (0's become 1's and 1's become 0's).

        Choosing a corner square (1,3,7,9) causes
the corner square and the three adjacent square to
change state.

        Finally, if you choose the center square (5),
all but the corner squares will change state.

        To end the game, hit <delete>.

 */

#include <stdio.h>

#define DEL 0x7f

int board[9];

/*
 *      main - perform one or more games
 */

main()
{
        int square;
        int win;
        int move;
```

```
        print_title();
again:
        setup_board();
        for (move = 1;;++move)
        {
                print_board(move);
                if (win = check_win()) break;
                for(;;)
                {
                 square = get_num("Your move");
                 if (square > 0 && square <= 9 && board[square-1])
                        break;
                 puts ("Illegal move--try again\n");
                }
                do_move(square);
        }
        switch(win)
        {
        case 1:
                puts ("You WON!\n");
                break;
        case 2:
                puts ("You LOST!\n");
                break;
        }
        if (get_yes("Would you like to try again")) goto again;
}

/*
 *       print_title -- print leading title.
 *       I was going to print a block title,
 *       except it'd look ugly...
 */

print_title()
{
        puts ("Brain teaser\n\n");
}

/*
 *       setup_board - create a random board.  Mostly zeros
 *       but at make sure at least one one
 */

setup_board()
{
        int i;
        int sum;
        unsigned seed;
        long time();

        sum = 0;
        seed = getpid();
        srand(seed);

        do for (i = 0; i < 9; ++i)
                sum += (board[i] = (rand() % 100) > 90);
        while (sum == 0);
}

/*
 *       print_board -- print the board out
 */

print_board(move)
int move;
{
        int i,j;

        printf("\nThe board after move %d\n\n",move);
        for (i = 0; i < 9; i += 3)
        {
                for (j = 0; j < 3; ++j)
                        printf("%d  ",board[i+j]);
                putchar('\n');
        }
        putchar('\n');

}

/*
 *       check for a win.  0 if not done yet, 1 if user
 *       wins, 2 if he looses.
 */

check_win()
{
        int s;
        int i;

        s = 0;
        for (i = 0; i < 9; ++i)
                s += board[i];
        if (board[4])
                s = 1;
        switch(s)
        {
        case 8:
                return 1;       /* win */
        case 0:
                return 2;       /* loss */
        default:
                return 0;       /* not done yet */
        }
}

/* state_change -- for do_move */

char state_change[] = {
        1,2,4,5,0,      1,2,3,0,0,      2,3,5,6,0,
        1,4,7,0,0,      2,4,5,6,8,      3,6,9,0,0,
        4,5,7,8,0,      7,8,9,0,0,      5,6,8,9,0,
};

/*
 *       do_move -- perform one move.
 */

do_move(where)
int where;
{
        int t;
        int i;

        --where;
        for (i = 0,where *= 5; i < 5; ++i,++where)
                if ((t = state_change[where]) == 0) break;
        else
        {
                --t;
                board[t] ^= 1;
        }
}

/*
 *       get_yes -- ask user a yes/no question.
 */

get_yes(msg)
char *msg;
{
        char line[5];

        for(;;)
        {
                puts(msg);
                puts("? ");
                if (!gets(line,5)) exit (0);
                if (*line == 'y' || *line == 'Y')
                        return 1;
                if (*line == 'n' || *line == 'N')
                        return 0;
                puts ("Please say yes or no\n");
        }
}

/*
 *       get_num -- get a number from the user.
 */

get_num(msg)
char *msg;
{
        int i;
        char c;

                puts(msg);
                puts("? ");
  do
   if ((c = getchar()) == EOL)
                        exit(0);
  while (c == '\n');
                return (i = c - '0');
}

/*
 *       getpid -- get a random number seed.
 */

getpid()
{
unsigned count;

        puts("Press any key\n");
        while(checkterm() == -1)
                ++count;
        return (count);
}
```

Basic OS-9

Ron Voigts

TICK, TOCK, TICK ...

I had the pleasure of getting my early experience on computers that were used for scientific experiments. I got to write programs, run other programs, build the hardware and set up the experiments. In all the data taking we did, there was one key element. One variable constantly occurred. It was time. There was elapsed time, and marked time. Time and appropriately the clock were an important part of the computer.

At the gut level all computers use time. Every action, every calculation, everything a computer does is timed. Take a look at the schematic for your computer and you'll see a little component identified as a clock or perhaps a crystal oscillator. Its purpose is to send out pulses to the the micro-P timing everything it does. It works like a little drummer pounding out time for the micro-P to march to.

There is another clock that the computer can use. It is the Real Time Clock or abbreviated, RTC. It is a separate clock that can be used to keep track of seconds, minutes, hours, days and months. The clock in my system can be read to a thousandth of a second. Some can can keep track of leap years. The RTC is important to OS-9.

On Level I the RTC is not necessary, but good to have. On Level II it becomes a necessary part of OS-9. It is so important to have a clock that Radio Schack created a clock module for the Color Computer that was run by the 6809, since a real hardware clock is not in the computer. As a side note the NMI line of the 6809 is connected to the RS Disk Controller. So when ever the drive is accessed the the 6809 is temporarily halted. The result is the that anyone running OS-9 on the Coco will notice that their clock is gradually losing time. The solution is to add a RTC that can be plugged into a Y-cable, or one of the expansion units that are plugged into the Coco's side port.

A real importance to having a RTC comes when you want to use OS-9's ability to multitask jobs. Let us say you would like list one file, while editing another. You would enter from the keyboard:

```
OS9:list file_1 >/p&
OS9:edit file_2
```

File_1 would be listed to the printer as a background task, while you worked on the second file. Time is allocated to the different tasks, based on the clock. The process is called 'timesliceing'. The CPU is interrupted by the RTC, usually 10 per second on Level I and 100 per second on Level II. Each interval is called a tick. The different processes are allocated a little time each second, so that everything appears to be running simultaneously. If it were not for the clock's interrupts this would not be possible. In fact if the clock has been not initialized, multitasking will not be possible.

An obvious use for the clock is to incorporate it into the OS-9 file structure. If you've done your homework and taken a look at your disk's structure as I suggested in the July and August 68' MICRO JOURNAL, you would know that the disk identification sector contains the date and time it was created. All file descriptor sectors contain the dete and time last modified and the date of creation. Without a clock OS-9 would load these areas with all zeroes. You would be losing a valuable piece of information in you file and disk structure.

The RTC comes in handy at the program level, too. When you use commands like DIR and FREE, they print the time along with the information that you request. The date and time gets incorporated into compiler listings like those created with the Microware Interactive Assembler and the Pascal Compiler. In fact you can use the date and time in your own programs. If you run Basic09, DATE$ returns the date and time in string form. With a little clever use of MID$ you should be able to extract the month, day, year, hour, minute and second from it. The Microware C language has a system call that can get the time or change it. They are **setime**(b) and **getime**(b), where b is a buffer defined by struct sgtbuf *b. This structure is located in time.h. The structure declaration looks something like:

```
struct sgtbuf {
   char  t_year,
         t_month,
         t_day,
         t_hour,
         t_minute,
         t_second;
};
```

So if you want the current hour you would put in your program, you would execute getime(b). The 'b' is a pointer to the date and time. The hour would be (*b).t_hour. (C Language has a shorthand version of this. The hour can be referenced also by b->t_hour. The '->' is shorthand for 'b' pointing to t_hour.) Finally, if your a Pascal programmer, a non_standard ISO procedure has been added. It is:

```
Procedure systime(year, month, day, hour, minute,
second: integer);
```

The concept is about the same as before. Using this call in a Pascal program will return the date and time.

At the heart of these calls are two OS-9 Service Requests Calls, **F$stime** and **F$time**. F$stime is used to set system time. The X register points to a 6 byte buffer which holds the year, month, day, hour, minute and second. It puts the new date and time into hardware and puts F$time via the 'install function request', F$ssvc. To put it into a nutshell, you can't get the time, unless you set it first.

At the OS-9 command level are two modules that permit you to use these two calls. They are **Setime** and Date. Setime has a number of ways you can use it. Some of them are :

```
OS9:setime 85,10,5 1730
OS9:setime 851005 173000
OS9:setime 85
OS9:setime
```

The first two variations set the date and time to October 5, 1985, 5:30 PM. The second version sets only the year. If you have a RTC that is battery powered, you'll want to use this version, since the other elements continue to update, even when your computer is off. The last version will prompt you for the necessary information. Usually you'll want to include Setime in your 'startup' file. If you have a real RTC with battery backup, you can use the 'setime 85' version. Color Computer users will have the last version, since their system must be reset at boot up. It is worth noting that even if you don't have any

clock in you aystem, you can use Setime to set the year, month and day. Then at least you files will be dated.

A NEW DATE FOR YOUR SYSTEM

I held off on the Date command until last. Its syntax is:

```
OS9:date
OS):date t
```

The first version prints the date and the second prints date and military time.

After some thought I decided to recreate Date. The original Date did its job, but I felt it had a few short comings. It first short coming is that it puts leading 0's into number smaller then 10. So you could get a date like October 06, 1985. (That is not how I would write it.) Secondly, the time is reported in military time. There is nothing wrong with reporting it this way. But when I talk to the rest of the world I use notations like AM and PM. There is something warmer about saying, it is 5:30 PM, instead of 1730 hours. Finally, I thought it a nice feature to report the day of the week. Occasionally seeing the date may not have as much impact as knowing what day it is.

I also decided to rewrite it as close as possible to the original, so the new version is syntactically the same as before. If you enter 'date' it will print the date and day. Entering 'date t' will get the date, day and time printed as an AM or PM.

Most of the program is relatively straight forward. I think if you read the comments and follow it carefully, you should not any trouble with it. Perhaps one part deserve some commentary. It is the computation of the day of the week. In past programs I had used a formula involving real numbers to find the day of week. Using real numbers in assembly language is a real challenge, so I opted to compute it using integer math. I fact the numbers involved use no more than a register or byte of memory. The formula I use is:

d=mod 7 of f+(y-(n-3)/4)+y-n

- d : day before Jan 1
- y : year
- f : normalizing factor
- n : normalizing year

In the above equation f is 60 and n is 4. If you apply the above formula for any year (i.e. for 1986, y=86), d will be the day before January 1st of that year. The program adds in the days in the previous months, adjusting for a leap year if necessary, and it adds in the days in the current month. Another mod 7 is taken of the sums and the current day is found. I have checked the formula for all days up to December 31, 1999 and it works. I am not sure after that time, but by then I will have a new version.

This version of 'date' is a little longer then the original, but I think you'll also find it a bit more friendly. That's it for now. See ya soon!

\- - -

```
0001      * THIS IS A REPLACEMENT FOR THE STANDARD OS-9
0002      * MODULE.  IT ACCEPTS "DATE" AND "DATE T" AS BEFORE.
0003      * THE MAJOR DIFFERENCES IS THE DAY OF THE WEEK IS
0004      * PRINTED WITH THE DATE.  IF THE "T" OPTION IS SELECTED
0005      * THE TIME IS PRINTED IN AM OR PM NOTATION, INSTEAD
0006      * OF MILITARY TIME.
0007      *
0008                        NAM   DATE
0009                        TTL   REPLACEMENT FOR STANDARD "DATE"
0010      *
0011      * BETWEEN IFP1 AND ENDC IS
0012      * USE /DD/DEFS/DEFSFILE
0013                        IFP1
0015                        ENDC
```

```
0016      *
0017      * MODULE NAME AND EQUATES
0018  0000 87CD822F           MOD   DATSIZ,DATNAM,TYPE,REVS,START,SIZE
0019  000D 446174E5 DATNAM    FCS   /Date/
0020  0020          SPACE     EQU   $20
0021  0004          NORMFAC   EQU   4          NORMALIZING FACTOR
0022  003C          NRMYR     EQU   60         NORMALIZING YEAR
0023  0020          BUFSIZ    EQU   32
0024  0011          TYPE      SET   PRGRM+OBJCT MOD TYPE
0025  0002          REVS      SET   REENT+2    MOD REVISION
0026      *
0027      * DATA AREA
0028 D 0000                     ORG   0
0029 D 0000          DATE      EQU   .          DATE BUFFER
0030 D 0000          YEAR      RMB   1
0031 D 0001          MONTH     RMB   1
0032 D 0002          DAY       RMB   1
0033 D 0003          HOUR      RMB   1
0034 D 0004          MINUTE    RMB   1
0035 D 0005          SECOND    RMB   1
0036 D 0006          TFLAG     RMB   1          TIME FLAG
0037 D 0007          PFLAG     RMB   1          PM FLAG
0038 D 0008          SMLBUF    RMB   2          SMALL BUFFER
0039 D 000A          BUFFER    RMB   BUFSIZ     PRINT BUFFER
0040 D 002A                    RMB   200        STACK AREA
0041 D 00F2                    RMB   200        PARAMETER AREA
0042 D 01BA          SIZE      EQU   .
0043      *
0044      * PROGRAM AREA
0045  0011          START     EQU   *
0046  0011 0F86               CLR   TFLAG      CLEAR TIME FLAG
0047  0013 0F87               CLR   PFLAG      CLEAR PM FLAG
0048      *
0049      * CHECK FOR T OPTION
0050  0015          GETOPT    EQU   *
0051  0015 A680               LDA   ,X+        GET A CHAR. FROM OPTIONS
0052  0017 810D               CMPA  #$0D       IS IT A C.R.?
0053  0019 2718               BEQ   GETDAT     YES, GO ON
0054  001B 8120               CMPA  #SPACE     IS IT A SPACE?
0055  001D 27F6               BEQ   GETOPT     YES, TRY AGAIN
0056  001F 8174               CMPA  #'t        IS IT A L.C. 'T'?
0057  0021 2706               BEQ   TSET       SET TIME FLAG!
0058  0023 8154               CMPA  #'T        IS IT A U.C. 'T'?
0059  0025 2702               BEQ   TSET       A TIME FLAG!
0060  0027 2002               BRA   GETDAT     TIME WON'T BE REPORTED
0061      *
0062  0029 0C06          TSET  COM   TFLAG      SET TIME FLAG
0063      *
0064      * GET THE CURRENT DATE AND TIME
0065  002B          GETDAT    EQU   *
0066  002B 30C4               LEAX  DATE,U
0067  002D 103F15             OS9   F$TIME
0068  0030 102500DB           LBCS  ERROR      GO TO "ERROR" IF C BIT SET
0069      *
0070      * WRITE DATE
0071  0034          WDATE     EQU   *
0072  0034 314A               LEAY  BUFFER,U   SET "Y" TO PRINT BUFFER
0073  0036 864A               LDA   #18        OFFSETS OF MONTHS
0074  0038 D601               LDB   MONTH      GET THE MONTH
0075  003A 5A                 DECB             ADJUST FOR MONTHS
0076  003B 3D                 MUL
0077  003C 308D0110           LEAX  MONTHS,PCR POINT TO START OF "MONTHS"
0078  0040 30BB               LEAX  D,X        POINT TO CURRENT MONTH
0079  0042 E680               LDB   ,X+        GET MONTH SIZE
0080  0044 170100             LBSR  XFER       PUT MONTH INTO BUFFER
0081  0047 170BF1             LBSR  PUTSPC     PUT IN A SPACE
0082  004A 9602               LDA   DAY        GET CALENDAR DAY
0083  004C 170B26             LBSR  TOSMBF     PUT IN SMALL BUFFER
0084  004F 9602               LDA   DAY        GET DAY AGAIN
0085  0051 8109               CMPA  #9         IS IT 9 OR LESS
0086  0053 2E04               BGT   WDT01      NO, THEN GO ON
0087  0055 8620               LDA   #SPACE     OTHERWISE PUT IN A SPACE
0088  0057 A748               STA   SMLBUF,U
0089  0059 170EE4        WDT01  LBSR  SMXFER     MOVE SMALL BUFFER
0090  005C 308D0170           LEAX  CNTURY,PCR GET THE CURRENT CENTURY
0091  0060 C604               LDB   #4
0092  0062 170EE2             LBSR  XFER       PUT IT IN BUFFER
```

```
0093 0065 9600            LDA   YEAR       GET THE YEAR
0094 0067 170088          LBSR  TOSMBF     PUT IT IN SMALL BUFFER
0095 006A 1700D3          LBSR  SMIFER     MOVE SMALL BUFFER
0096 006D 170080          LBSR  CR         ADD A CARRIAGE RETURN
0097 0070 170095          LBSR  WRITIT     WRITE THE THE DATE
0098                      *
0099                      * WRITE DAY OF WEEK
0100 0073           WDAY  EQU   *
0101 0073 9688            LDA   YEAR       GET THE YEAR
0102 0075 8039            SUBA  #NRMYR-3   SUBTRACT NRMLIZING YEAR-3
0103 0077 44              LSRA             DIVIDE BY 4
0104 0078 44              LSRA
0105 0079 9000            ADDA  YEAR       ADD YEAR TO IT
0106 007B 8B04            ADDA  #NRMFAC    ADD NORMALIZING FACTOR
0107 007D 803C            SUBA  #NRMYR     SUBTRACT NORMALIZING YEAR
0108 007F 1700CD          LBSR  MOD7       DO A MOD 7
0109 0082 D681            LDB   MONTH      GET THE MONTH
0110 0084 308D0192        LEAX  MSIZE,PCR  ADD DAYS OF PREVIOUS MONTHS
0111 0088 5A        DW01  DECB
0112 0089 2704            BEQ   DW02
0113 008B A880            ADDA  ,X+
0114 008D 20F9            BRA   DW01
0115 008F D688      DW02  LDB   YEAR       GET YEAR
0116 0091 C503            BITB  #1000B0011 IS IT LEAP YEAR?
0117 0093 2607            BNE   DW03       NO THEN GO ON
0118 0095 D681            LDB   MONTH      GET THE MONTH
0119 0097 C103            CMPB  #3         IS IT MARCH OR LATER?
0120 0099 2501            BLO   DW03       NO THEN GO ON
0121 009B 4C              INCA             ADD 1 FOR THE LEAP YEAR
0122 009C 9B02      DW03  ADDA  DAY        FINALLY ADD IN THE DAYS
0123 009E 1700AE          LBSR  MOD7       DO A MOD 7 AGAIN
0124 00A1 314A            LEAY  BUFFER,U   POINT TO BUFFER
0125 00A3 308D0120        LEAX  DAYS,PCR   POINT TO DAYS BUFFER
0126 00A7 C6DA            LDB   #19
0127 00A9 3D              MUL              ADJUST FOR RIGHT DAY
0128 00AA 308B            LEAX  D,X
0129 00AC E680            LDB   ,X+
0130 00AE 170096          LBSR  XFER       PUT DAY INTO BUFFER
0131 00B1 8D6D            BSR   CR         ADD A CARRIAGE RETURN
0132 00B3 8D5D            BSR   WRITIT     AND WRITE IT
0133                      *
0134                      * WRITE THE TIME
0135 00B5           WTIME EQU   *
0136 00B5 9686            LDA   TFLAG      SEE IF WE WANT THE TIME
0137 00B7 4D              TSTA
0138 00B8 2754            BEQ   NOERR      NO, THEN END
0139 00BA 9603            LDA   HOUR       GET THE HOUR IN M.T.
0140 00BC B10C            CMPA  #12        IS IT AM?
0141 00BE 250A            BLO   WTM01      YES THEN GO ON
0142 00C0 0307            COM   PFLAG      OTHERWISE SET PM FLAG
0143 00C2 810C            CMPA  #12        IS IT 12 PM?
0144 00C4 270A            BEQ   WTM02      YES THEN LEAVE IT ALONE
0145 00C6 800C            SUBA  #12        ELSE ADJUST IT FOR  PM TIME
0146 00C8 2006            BRA   WTM02
0147 00CA 8100      WTM01 CMPA  #0         IS IT 00 HRS?
0148 00CC 2602            BNE   WTM02      NO, GO ON
0149 00CE 860C            LDA   #12        ELSE IT IS 12 AM
0150 00D0 314A      WTM02 LEAY  BUFFER,U   POINT TO THE BUFFER AGAIN
0151 00D2 8D51            BSR   TOSMBF     PUT HOUR INTO BUFFER
0152 00D4 9600            LDA   SMLBUF     CHECK SMALL BUFFER
0153 00D6 8130            CMPA  #'0        IS A LEADING 0
0154 00D8 2604            BNE   WTM03      NO, THEN GO ON
0155 00DA 8620            LDA   #SPACE
0156 00DC 9700            STA   SMLBUF     CHANGE IT TO A SPACE
0157 00DE 8D60      WTM03 BSR   SMIFER     MOVE SMALL BUFFER
0158 00E0 863A            LDA   #':        ADD TIME SEPERATOR
0159 00E2 A7A0            STA   ,Y+
0160 00E4 9604            LDA   MINUTE     GET MINUTES
0161 00E6 8D3D            BSR   TOSMBF     PUT MINUTES IN SMALL BUFFER
0162 00E8 8D56            BSR   SMIFER     MOVE SMALL BUFFER
0163 00EA 863A            LDA   #':        ANOTHER SEPERATOR
0164 00EC A7A0            STA   ,Y+
0165 00EE 9605            LDA   SECOND     GET SECONDS
0166 00F0 8D33            BSR   TOSMBF     PUT IT IN SMALL BUFFER
0167 00F2 8D4C            BSR   SMIFER     MOVE SMALL BUFFER
0168 00F4 0D07            TST   PFLAG      IS IT PM?
0169 00F6 270A            BEQ   AM         NO, GO TO AM
0170 00F8 308D012D  PM    LEAX  PMT,PCR
0171 00FC C603            LDB   #3
0172 00FE 8D47            BSR   XFER       PUT PM IN BUFFER
0173 0100 2008            BRA   WTM04
0174 0102 308D0120  AM    LEAX  AMT,PCR
0175 0106 C603            LDB   #3
0176 0108 8D3D            BSR   XFER       PUT AM IN BUFFER
0177 010A 8D14      WTM04 BSR   CR         ADD A CARRAIGE RETURN
0178 010C 8D04            BSR   WRITIT     WRITE IT
0179                      * END OF PROGRAM
0180 010E 5F        NOERR CLRB             CLR B IF NO ERRORS
0181 010F 103F06    ERROR OS9   F$EXIT     FINISH UP
0182                      *
0183                      * WRITE BUFFER OR UP TO CARIAGE RETURN
0184 0112 8681      WRITIT LDA  #1         STANDARD OUPUT
0185 0114 304A            LEAX  BUFFER,U   BUFFER LOCATION
0186 0116 108E0020        LDY   #BUFSIZ    BUFFER SIZE
0187 011A 103F8C          OS9   I$WRITLN   WRITE IT
0188 011D 25F0            BCS   ERROR
0189 011F 39              RTS
0190                      *
0191                      * PUT IN A CARRIAGE RETURN
0192 0120 860D      CR    LDA   #$0D       GET C.R. CHARACTER
0193 0122 A7A4            STA   ,Y         STORE IT IN BUFFER
0194 0124 39              RTS
0195                      *
0196                      * CHANGE "A" TO BCD FORM
0197 0125           TOSMBF EQU  *
0198 0125 6F48            CLR   SMLBUF,U   CLR FIRST PART OF BUFFER
0199 0127 810A      0001  CMPA  #10        IS "A" SMALLER THAN 10?
0200 0129 2506            BLO   0002       YES THEN GO ON
0201 012B 800A            SUBA  #10        OTHERWISE SUBTRACT 10
0202 012D 6C48            INC   SMLBUF,U   INCREMENT SMALL BUFFER
0203 012F 20F6            BRA   0001       TRY IT AGAIN!
0204 0131 A749      0002  STA   SMLBUF+1,U SAVE THE BALANCE OF A
0205 0133 EC48            LDD   SMLBUF,U
0206 0135 C3303D          ADDD  #$303D     CHANGE TO ASCII FORMAT
0207 0138 ED48            STD   SMLBUF,U
0208 013A 39              RTS
0209                      *
0210                      * PUT A SPACE INTO THE BUFFER
0211 013B 8620      PUTSPC LDA  #SPACE     GET SPACE
0212 013D A7A0            STA   ,Y+        PUT IT IN BUFFER
0213 013F 39              RTS
0214                      *
0215                      * SMALL BUFFER DATA TRANSFER
0216 0140 3048      SMIFER LEAX SMLBUF,U   POINT TO SMALL BUFFER
0217 0142 C602            LDB   #2         MOVING 2 BYTES
0218 0144 8D01            BSR   XFER
0219 0146 39              RTS
0220                      *
0221                      * TRANSFER DATA TO BUFFER
0222 0147 A680      XFER  LDA   ,X+        GET A CHARACTER
0223 0149 A7A0            STA   ,Y+        PUT IN BUFFER
0224 014B 5A              DECB             DO IT "B" TIMES
0225 014C 26F9            BNE   XFER
0226 014E 39              RTS
0227                      *
0228                      * FIND MOD 7 OF "A"
0229 014F 8107      MOD7  CMPA  #7         IS "A" LESS THAN 7
0230 0151 2504            BLO   MOD1       YES THEN GO ON
0231 0153 8007            SUBA  #7         ELSE SUBTRACT
0232 0155 20F8            BRA   MOD7       AND TRY AGAIN
0233 0157 39        MOD1  RTS
0234                      *
0235                      * TABLES AND CONSTANTS
0236                      *
0237                      * MONTHS
0238 0158           MONTHS EQU  *          MONTH TABLE
0239 0158 07                  FCB   7
0240 0159 4A616E75            FCC   /January /
0241 0162 08                  FCB   8
0242 0163 46656272            FCC   /February /
0243 016C 05                  FCB   5
0244 0169 4D617263            FCC   /March   /
0245 0176 05                  FCB   5
0246 0177 41707269            FCC   /April   /
0247 0180 03                  FCB   3
0248 0181 4D617920            FCC   /May     /
```

```
00249  018A 04              FCB  4
00250  018B 44756E65        FCC  /June    /
00251  0194 04              FCB  4
00252  0195 4A756C79        FCC  /July    /
00253  019E 06              FCB  6
00254  019F 41756775        FCC  /August  /
00255  01A8 09              FCB  9
00256  01A9 53657074        FCC  /September/
00257  01B2 07              FCB  7
00258  01B3 4F63746F        FCC  /October /
00259  01BC 08              FCB  8
00260  01BD 4E6F7665        FCC  /November /
00261  01C6 08              FCB  8
00262  01C7 44656365        FCC  /December /
00263        *
00264        * THE CURRENT CENTURY
00265  01D0 2C203139  CNTURY  FCC  /, 19/
00266        *
00267        * THE DAYS OF THE WEEK
00268  0104           DAYS    EQU  *         DAYS TABLE
00269  0104 06              FCB  6
00270  0105 53756E64        FCC  /Sunday  /
00271  010E 06              FCB  6
00272  010F 4D6F6E64        FCC  /Monday  /
00273  01EB 07              FCB  7
00274  01E9 54756573        FCC  /Tuesday /
00275  01F2 09              FCB  9
00276  01F3 5765646E        FCC  /Wednesday/
00277  01FC 08              FCB  8
00278  01FD 54687572        FCC  /Thursday /
00279  0206 06              FCB  6
00280  0207 46726964        FCC  /Friday  /
00281  0210 08              FCB  8
00282  0211 53617475        FCC  /Saturday /
00283        *
00284        * ADJUST FOR PREVIOUS MONTHS
00285  021A 030003    MSIZE   FCB  3,0,3    MONTHS SIZE MINUS 28
00286  021D 020302            FCB  2,3,2
00287  0220 030302            FCB  3,3,2
00288  0223 030203            FCB  3,2,3
00289        *
00290        * AM AND PM NOTATION
00291  0226 204140    AMT     FCC  / AM/
00292  0229 205040    PMT     FCC  / PM/
00293  022C 011926            FCB  EMOD
00294  022F          DATSIZ   EQU  *
00295                 END
```

OS-9 User Notes

Peter Dibble
19 Fountain Street
Rochester, NY 14620

Microware Seminar

I am freshly back from this year's OS-9 Seminar. Fun! This year the exhibit hall was the most interesting part of the seminar. Single-board computers were very much the thing. I only saw a single S-50 machine there. Likewise, 68Ks have replaced 6809's. I think the only 6809 in use was at the Seiku booth where there were Fujitsu 6809 and 68K computers. No, I take that back, there were a few more: a time- card system, a CoCo, and a Smoke Signal VAR. Still, the place was full of 68K systems in cases with Plexiglass tops.

Gimix was holding the high ground with their GMX-20 board. It was almost still warm from the PC-board manufacturer. I, and I think everyone else, was impressed with its performance.

The C compiler is my favorite bench mark. I spend more time waiting for the compiler to run than any other computer operation. I time the compiler on the GMX-20. I was able to get a small C program through the entire C cycle (preprocessor, compiler, optimizer, assembler, and linker) in about three seconds. Kim Kempf used pipes and did the same thing in about half the time. That's fast!

The price Gimix wants for it is, in my judgment, too high. It is a fine board and many people will be happy to get it for $2750, but by the time it is integrated into a system the cost will more like $5000. I am a computer nut; I will sometimes pay a lot of money for extra power -- even useless power. The problem is that most of the time the power is invisible. Who cares how fast the computer waits for characters while you edit? Gimix will probably sell all they can build, but a lot of potential customers just won't be able to handle the price for the small additional usefulness.

I thought I'd come back from the seminar with my mind made up about what 68K system to get, but I didn't. The Gimix is too expensive for a machine that will be outclassed in a year. The other computers have fine prices but the performance difference is too much to ignore. I also feel nervous about computers that don't have a buss. How am I supposed to add memory to these machines? Maybe I'll grit my teeth and wait for a computer that is an obvious choice.

Seiku had lots of nice things to show. Three of them were particularly interesting. They have managed somehow to run CP/M as a task under OS-9. I ran WordStar under CP/M in a window while other things were going on under OS-9. As they put it "OS-9 is a wonderful operating system but it is very short on application software. CP/M is a bad operating system with plenty of application software. We have combined them."

Even more exciting to me was the Trina/Serina combination. They are a sophisticated application debugger and a powerful debugger for system software.

The system debugger needs to have code written to allow it to reach the terminal (and, optionally, the printer) without using OS-9 but it can be used to debug drivers and even kernal code.

This seems to be the season for OS-9 debuggers. Several good ones are on their way to market but this is special. The Erina/Serina combination finally gives us a way to debug system software. I hope they can sell them in this country.

The new-products session was the big event Sunday.

Microware's Fortran compiler didn't quite make it to the seminar. It is still very close but not quite here. I spoke to several people with pre-release versions of it trying to get some feel for how close it was. The people who had played with it a lot seemed very pleased with it. Those who had just tried a few test programs were having trouble making it work. Based on that, I'd guess that Microware is working on documentation and superficial bugs.

We saw the OS-9 networking package demonstrated. It looks like a good job, but I hope they find a way to give us aliases for path names (other than the "." names). To access a file on another machine you type: /net/<nodename>/<device>/<dir>.../file. You can change directories if you like, but typing in pathlists for other machines could get boring very fast.

The networking idea fits smoothly into OS-9. There is a network file server which handles most of the protocol. I think device drivers for the network are expected to be able to send and receive packets. Any I/O hardware that can handle inter-machine communications can be used. (The system they demonstrated was connected using Arcnet and ran fast.)

There were two glitches that effected the new-products session. They did more than give the seminar a human feel; they were positively funny. I guess it's a little untactful for me to emphasis them, but Microware is running the seminar more smoothly each year. I like the slightly disorganised family atmosphere that has been fading out of the seminar, so when some of it appears I cherish it. For the record this seminar was another example of smooth organization.

On Friday and Saturday eveyone who was interested could see the network running between three machines in two different booths in the exhibit hall. Sunday morning, when the Microware staff went to set the network up in the lecture hall for their demonstration they found that the coaxial cables for the network had been mangled. It looked like they had been run over with a lawn mower. Is it possible? They were able to salvage enough wire to connect two processors sitting right next to each other. The demonstration was fine, but we were called on to believe that it would work with more than two machines.

Have you ever had bad dreams about being suddenly called upon to give a speech or demonstration without being prepared. It happened to one Microware programmer who will remain nameless. He was told shortly before the new products session that he would be showing the new Pascal compiler for the 68K. He did pretty well considering that he seemed to have mostly forgotten how to use the compiler he was responsible for. We were all rooting for him.

The presentation of new graphics software for OS-9 went off without a hitch. I have been pretty hardened to flashy graphics demonstrations by the expensive toys we have at school, but I was surprised at the speed and power of the demonstrations system. It drew the three-D curve usually called the cowboy hat much faster than an IBM PC does. I think several hundred times faster.

The Hitachi chip that the graphics software supports includes window support but Ken Kaplan wouldn't state Microware's intentions about windows. Let's all hope that Microware notices that windows are the "thing" and starts to support them in some hardware independent way.

A new release of OS-9 for the 6809 is coming out early next year. It will be available first for the CoCo. Microware wasn't very specific about what would be included in the new release, but there were some hints. They were using a Radio Shack hard disk attached to their CoCo. The new OS-9 should support a hard disk if Tandy decides to sell the interface card. They also had some kind of add-on that made the CoCo produce 80-column output with lower case letters. The new release of OS-9 will notice the wider screen and use it. It will also support networking if the network file manager is added (or by some chance included).

Microware has written a utility which can be used to "tune" the serial port. It adjusts the timing constant for the port. You are supposed to tweak it until whatever you have attached works or you have the baud rate you are looking for.

They have better graphics support. Since the new graphics support module is larger than the old one Microware has broken it in two. There is a module that only provides simple graphics and another that handles the more complicated calls. You won't need to waste space on software for region flooding if you don't plan to use it.

They have carried the philosophy of allowing users to select from a large menu of OS-9 features through the package. I think they said that you can even run your CoCo without CCIO.

Nothing was said about features and utilities from the 68K being moved to the 6809.

The Japanese contingent at the seminar has always impressed me with their enthusiasm. This year there was a session called The Japanese Connection. I had to leave part way through but from what I heard it was clear to me that selling OS-9 in Japan started as a labor of love and has turned into a great success.

I hope I'll see you all at the seminar next year.

The Complete Rainbow Guide to OS-9 by Dale Puckett and myself has been bought by Tandy. You should be able to buy as many copies as you like at your local Radio Shack. It looks like you'll still have to get the disks that go with the book from Rainbow Magazine. My User Notes book has been published in Japan -- in Japanese! I am flattered.

I'm getting married in about a month. You will all appreciate this because she is a great proof reader.

- - -

68000 User Notes

Phillip Lucido
2320 Saratoga Drive
Sharpsville, PA 16150

In the last column, I gave you a small program, whose purpose was to acquaint you with various aspects of assembly language programming. As normally happens, I ran out of space before telling you how to run the program. This time around, I'll describe how to get from the example program, in its text form, to something the computer can understand, and then how to watch the computer do its thing on the program.

Turning Code Into Mush

Computers (as I'm sure you've discovered by now) are extremely picky when it comes to feeding them. You must be careful to type everything just so, with everything in its proper place, and nothing forgotten. Programming in assembly language is certainly no exception. If anything, it shows computers in the worst light possible. Usually, if you manage to do something wrong, the computer will simply complain a little, and ask you to try again. Here, if you get something out of order, or don't do all the necessary steps, the computer may decide to go on strike, and refuse to acknowledge your existence. (Do I anthropomorphize a bit too much here? Sorry - a risk you take when you work with computers as much as I do.)

First off, you've got to put the program into the computer. If you haven't done so already, type the 18 lines of the test program (see last month's column), into a file named **prog001.a**. Be very careful to type the program in as shown. A few things are not that important, such as the blank lines or the comments, but other parts must be exact.

Now, as I mentioned last month, a source file like you've just entered isn't suitable for consumption by the computer. First, it must be properly sliced, diced, and mashed, or, in our case, assembled, linked, and debugged. The exact mechanics of the assembly process vary widely between operating systems, so what I say applies directly only to OS-9/68K. The broad details, though, are common to most computers.

The first step in reducing your source file to its final form is the assembler. This will produce a relocatable object file. Type in the following command line to perform this step:

```
r68 prog001.a -o=prog001.o
```

The relocatable object file, **prog001.o**, holds everything of interest to the computer which occurs in the source file. It is not yet an executable program, though. As I mentioned last time, multiple relocatable object files can be combined, along with common routines from library files, to result in the final executable program. This is done with the linker, which requires some extra information from the assembler to combine the files properly. In the current case, there is nothing to be combined, but we still have to run the linker, to get rid of the extra information and arrive at an executable program. The following command line invokes the linker:

```
l68 prog001.o -o=prog001 -g
```

The final executable program is prog001. This contains everything needed by the operating system to actually run the program. In addition to the numbers corresponding to the instructions in the original source file, the executable file includes information about the size, type, and name of the program. This is required by OS-9, so that many programs can comfortably reside in memory together. We won't have any use for the extra information in the near future, though, so I'll just ignore it for now.

Note the -g option to the linker. This causes a file named prog001.stb to be created in addition to the executable file. The .stb file records the address information for all global symbols, which include the labels in the source text which were used with colons following them. This allows symbolic debugging, which I'll get to next. For now, just remember that without the -g, the .stb file will not be created.

When you run the assembler and linker, you may get some error messages. If you do, go back and check the source file carefully. Since this example uses no library files, this is the only explanation (other than system failure) for errors.

Debugging Your Way To Enlightenment

If you've followed the above steps properly, your executable file can be run with the command "prog001". If you try this, though, the machine will seem to lock up. This is because the program doesn't produce any output, and doesn't exit when it is finished. To properly use the program, you need the debugger.

Execute the command "debug prog001". The debugger should start, and will read the files prog001 and prog001.stb from the execution directory. The debugger now knows about your program, and also knows where the various labels are located within it.

Let's start debugging by examining the original source code, as seen by the debugger. In response to the prompt "dbg:", type the command "di start 7", and you will see the following:

```
start     >303C0000     move.w #0,d0
start+4   >323C0001     move.w #1,d1
loop      >D041         add.w d1,d0
loop+2    >D27C0001     add.w #1,d1
loop+6    >827C0064     cmp.w #100,d1
loop+A    >6FF4         ble.s loop
done      >60FE         bra.s done
```

What you've just done is ask the debugger to disassemble the 7 lines of your program, beginning with the line that has "start:" at the front of it. To disassemble a program is to take the numbers which are a program to a computer, and turn them into some sort of text which looks like an assembler source file. As such, it is the opposite of assembling.

The disassembly consists of three columns. The middle column holds the actual numbers being disassembled, in hexadecimal. The first column is the address of each line in memory. Those lines in the original program that began with a label, like "start", "loop", or "done", have the same name as the address. Those lines that did not begin with a name in column 1 have an address which is the nearest previous label plus a number (in hex) which is the number of bytes between the label and the start of the line in question.

The third column of the disassembly is the actual textual representation of a single instruction. Compare the third column with the original source text. They are almost identical. The main difference is the addition of ".w" to many of the opcodes. In the last column, I mentioned that most 68000 opcodes can operate on 8 bit (byte), 16 bit (word), or 32 bit (longword) data. A particular data size is chosen by putting one of the suffixes ".b", ".w", or ".l" after the opcode. To save typing, though, the assembler allows you to omit the size suffix, in which case a default of ".w" is normally assumed. The debugger, on the other hand, goes ahead and puts the size suffix on just about everything.

Back to operating the debugger. Notice that the prompt is now "dis:". After certain operations, just pressing return on an empty line will cause the last operation to be continued. For instance, pressing return now will cause the next 7 lines to be disassembled. Since there aren't any more lines in the program, though, what is disassembled is not the program, but more or less random data.

Instead of disassembling more, try the command "s". This will print the symbol table, which should look something like the following, though in three columns, not two, and with different numbers:

_jmptbl	D 0000C200	end	D 0000C200
btext	C 000F2F00	bname	C 000F2F48
start	C 000F2F50	loop	C 000F2F58
done	C 000F2F64	etext	C 000F2F66

Remember from last month that a symbol is just a name used to refer to a particular location in memory, which allows instructions and data to be referenced without worrying about the actual memory address. What the "s" command has printed is a list of all the global symbols in the program, along with the addresses at which they ended up. This data has been retrieved from the .stb file mentioned above, prog001.stb in this case.

First, some information. A global symbol is a symbol which has been made known by the assembler to the linker. These are the labels which were used with a colon following them. It is possible to use a label without the colon, but then the assembler would not put the symbol's name in the relocatable object file, and the linker would not be able to record the symbol data in the .stb file. Since very large programs generally have huge numbers of labels, it is a good idea to only make the most important symbols global. For now, though, we'll continue making everything global, so all the labels will show up in the debugger.

Examine the symbol table above. There are three fields for each entry. First, there is the name of the symbol. Next, there is a letter code, a D or a C in this case. Finally, there is an 8 digit hexadecimal number, which is the address corresponding to the symbol. Also, notice that in addition to the labels "start", "loop", and "done", which were defined in prog001, there are a number of additional symbols, like "end" and "bname". The letter code and the extra symbols have to do with that extra information that is part of an executable program under OS-9, which I'm ignoring for now. It's not that important to us now; just don't be surprised when symbols that aren't your own show up in your code.

You won't really have much need for the data in the symbol table. Instead, it is there for the debugger, so you can keep referring to places in memory by name, not number. When you typed the command "di start 7", the debugger noticed that "start" is a symbol, and looked for it in the symbol table. In my case, it found the value $000F2F50, and proceeded to disassemble the data starting at this address in memory. This is automatic, and much more convenient then trying to keep track of all those addresses yourself.

All This To Get The Answer 5050?

Enough playing around. Let's run this thing! First, type the command "." (a period). You'll get a list something like the following, though not split up into five lines, just three:

```
Dn: 00000005 00000000 00000080 00000003
    00000000 00000001 00000300 00000000
An: 00000000 0000C500 00000000 000F6900
    00000000 0000C4FE 00014200 0000C4FE
PC: start   >303C0000    move.w #0,d0    CC: -----
```

This is a register dump. Remember that the 68000 has 16 general registers, named D0 to D7 and A0 to A7. The first line of the register dump gives the value of the eight data registers D0 to D7. The next line displays the eight address registers A0 to A7. The third line of the register dump looks much like a line from the disassembly. PC is the name of another register, the Program Counter, which is the address of the next instruction to be executed. Here, the PC holds the address of the line labelled "start", and a disassembly of the instruction follows. Finally, the CC: ----- is the current contents of the Condition Codes register. This is the register used in conditional branch instructions. I'll explain it fully in a later column.

When debug is started with a program to be debugged, it starts out ready to execute the first instruction in the program. Thus, the register dump shows the debugger waiting to execute the "start" line from our little test program. Note that the registers shown here are as they exist just prior to executing the instruction shown. Also note that there are nonzero values in several registers. These are values set up by OS-9, to give information to the program about various matters, such as the command line and the program location. None of these is important for now.

Now enter the instruction "t". This will cause a single instruction to be executed, or "traced". After the line is executed, another register dump appears. Notice that this line shows a value for D0 of $00000000, or zero. Since the instruction just executed (from the previous register dump) was "move #0,d0", which puts a zero in D0, this makes good sense. The current register dump shows a disassembled instruction of "move #1,d1", which is the next instruction to be executed. The displayed D1 value is not 1, though, since the instruction has not yet been executed.

Type return, without anything else on the line. In the current mode, this will default to tracing a single instruction. Another register dump appears, this time with a value of 1 for D1. Keep pressing return, and study the register dump that results each time. After the "move #1,d1" comes "add.w d1,d0", which results in a 1 in D0 (1 + 0 = 1). Next, "add.w #1,d1" puts a 2 in D1. The compare instruction, "cmp.w #100,d1" does not change any of the general registers, but note that the CC register changes after the compare is executed. At the end of the loop, the conditional branch is displayed as "ble,s loop->", at least in my copy of the debugger. The "->" indicates that the branch is to be taken. Press return again, and you will see that the next instruction to be executed is back at label "loop".

Keep pressing return, and watch as the number in D1 increments by 1 each time through the four line loop, while the value in D0 goes from 0, to 1, to 3, to 6, and so on, keeping track of the subtotal so far.

As you can see, tracing and examining register dumps allows you to keep careful track of exactly what is going on. Unfortunately, the loop in this program will execute 100 times, or 400 instructions in all. This is a lot of time spent pressing return. There is, of course, a better way.

Type the instruction "b done", and follow it with the instruction "g". The first of these sets a breakpoint, which is a message to the debugger to stop when a certain point in the program is reached. Here, the debugger is told to stop when it gets to the line labelled "done", which is right after the add loop finishes. The "g" instruction just means to go, that is, start executing instructions without pausing and register dumping after each line. The execution continues until the breakpoint is reached, at which time the program stops and another register dump is printed.

The program is now done! But, did it work? Well, the final register dump shows $000013BA in D0, and $00000065 in D1. Type "v .d0" and "v .d1". The "V" command just prints the value of an expression in both hexadecimal and decimal. ".D0" means to take the value currently in D0, and similarly for ".d1". The value of D0 is 5050 decimal, and D1 holds 101 decimal. You might remember from way back in algebra that the sum of all numbers from 1 up to N is (N*(N+1))/2, or in this case, 100*101/2 = 5050, so the program seems to have worked.

Good enough! Type "q" to quit and return to OS-9. Reenter the debugger, if you like, and read the manual and try out some of the other commands. I'll say more about the debugger later, as we get to use more of its capabilities, but by now it should be obvious that a powerful debugger will greatly assist you in delving into the intricacies of assembly language. It is hard to overestimate the value of watching a program execute line by line, and being able to see it unfold as it modifies registers and memory.

HOW TO DO A WINCHESTER

By: Mickey Ferguson
POB 87
Kingston Springs, TN 37082

I think I'm in trouble! I think that I am in trouble with my wife. And I know that I am in trouble with DMW and the good people at 68 Micro. The reason for my difficulties is that I have spent every spare moment for the past few months where I am now; sitting in front of this terminal doing "stuff" with my computer. You see I first built this machine ten years ago when it started life as a SWTPC 6800 system. It then had 16k of memory, a CT-1024 terminal, an AC-30 cassette interface, and a PR-40 printer. It was a truly "state of the art" system. Now the state of the art is not a static thing, and my old 'puter is generally considered something from the dark ages. Even tho it has grown and evolved over the years to the point of having floppy disk drives, digital tape drives, 6809 CPU, hardware number cruncher, separate 6802 IO FEP, etc. But in the past few months, it has become a whole new system (or so it seems).

Here is what happened. In January 1985 I received a flyer in the mail from Priority One Electronics offering new Shugart 604 Winchester drives for $99 each. These were brand new drives in factory sealed boxes, tho only 5meg each. Well I did not know how I would use them, but no ham radio operator can resist this kind of bargain, so I ordered a couple. And also ordered a case with power supply for them. Then I started looking thru all my back issues of various computer magazines, paying particular attention to all the tiny little ads in the back. You know, the ones you have to use a microscope to read! Then armed with this information, I began making phone calls until I found the best possible price on an XEBEC S1410 hard disk controller. But you must understand, that was the best possible price on that particular day. As you know, this varies almost daily! When it arrived, the controller was XEBEC's S1410A, which is an improved version with additional features and half the power consumption of the original.

Having gotten all of this good stuff together; I then began searching back issues of the magazines again. But this time shopping for information on how to put the controller and drives to use on my system. As it turned out, the most valuable magazines were 68 Micro and Byte (in that order) while all the others were useless. [The following is an editorial type comment that 68 Micro will probably feel they need to delete.] It was revolting going thru other magazines looking for information of this nature. Most seem to feel the way to add a hard disk system to an existing computer is, "Be sure to have your purchasing agent specify....". What has happened to those old hackers, like me, who actually like to BUILD things? [End of editorial comment.]** The main thing I learned from the manuals on the drives, hard disk controller, and the magazines can best be summed up this way. I needed to do three things to get the drives working on an SS-50 system, first, I needed to scrounge a bunch of ribbon cable and connectors to tie everything together. Second, I needed to wire-wrap a PIA based host adaptor to convert SASI to SS-30, and finally do the software to tie it into Flex.

If you have had no experience with hard disk drives, of the five inch variety, some of the terms I have used may seem a little strange. So allow me to explain how they are implemented. All five inch winchesters are identical at their ribbon cable (and power) connectors; just as all floppys are. But the hard disks are very different from floppys! This interface is called by several different names, but all of them have Seagate in the name of the definition. You see, Seagate Technologies developed it and deserves the credit. All five inch Winnie controllers are designed to connect to this type of drive. The computer end of the hard disk controller is normally called SASI (pronounced sassy, like a smart-mouthed child). SASI is short for Shugart Associates Standard Interface. The host adaptor converts the signals from your particular

computer to SASI and from SASI to those your computer likes. This lets your computer deal with the hard disks (electrically) much like a parallel printer when outputting and parallel keyboard when inputting. The XEBEC controller is an industry standard, in my opinion. And is used in quite a number of hard disk systems from a wide variety of sources for numerous computers. In many ways it is to hard disk systems what D.C. Hayes is to smart modems. Almost, but not quite! It has its own processor, RAM, sector buffers, and command language. Allowing data transfers one byte at a time with full handshaking, to or from the hard disk. It also has diagnostics for both itself and the drives. The XEBEC controller is so smart that you could ALMOST use it to put Winchesters on an ADM3, Apple II or other dumb terminal!

Having acquired all of this useful information and being ready to start putting it into practice; I was immediately side-tracked for several months by installing paneling, laying ceramic tile, doing electrical wiring, etc. Do ALL WIVES look at unfinished basements as extensions of living space or is it only Foxy (my wife)???

By the time all of the blisters, smashed thumbs, and so forth had healed and I was again ready to tackle the hard disk project. A new issue of 68 Micro arrived in the mail, and this one contained an ad by WellWritten Enterprises for their host adaptor, software, and complete systems to add hard disk drives to SS-50 computers. So out comes the microscope and I find that they are intended for use with the XEBEC controller! I thought that $200 for the host adaptor seemed a bit high. But it did come with the software on disk to tie it into either Flex or OS9 whichever you choose. Well, I may be cheap, but I am also lazy. And if the software could be had without having to do it myself, and if Wellwritten Enterprises could be persuaded to provide all the necessary ribbon cables; it just might be worth it! So I picked up the phone and called the number in the ad....it was answered by a real live human named Tom Weaver! He turned out to be a person much like myself (what a relief!)....We also have an AT&T PC6300 and have been dealing quite a bit with AT&T Information Systems which is staffed entirely by people who are not able to walk and chew gum!!! Anyway Tom and I had a nice long chat during which we discovered that we both were hackers; we cried on each other's shoulders about the problems we have had with IBM PC's (and/or clones) and the need to have them. And many other things, much to the detriment of my poor old phone bill! The call ended with my placing an order for WellWritten's host adaptor and Flex software. And Tom agreeing to modify the software for TWO hard drives (no one had ever requested THAT before) and to include all the necessary ribbon cables for an additional fifty bucks. That might seem like a rather high price for the cables, but I had been checking to see what it was going to cost to buy the necessary cable and connectors; so it seemed like a bargain to me!

When Tom and I hung up, I immediately placed a call to DMW and suggested that I do this article. (Ouch! There goes the phone bill again!) I must have been persuasive because he agreed, reluctantly, to use it assuming it had redeeming social value. And I PROMISED to have it to him within a month (several months ago). Now you see why I am in trouble with 68 Micro!

Within a week after the phone call to WellWritten, I received in the mail the host adaptor, cables and software. And a letter from Tom filled with profuse apologies for the delay which he said were caused by the time required to modify their software drivers for my needs. Actually, I was pleased as this was the best turn-around time ever on anything I had ordered for any of our computers. But please don't tell Tom cause he seemed to feel really bad about the delay!

WellWritten's software was close to what I needed but, as always, not quite what I wanted. Their drivers are 100% relocatable and when installed move themselves to the top end of memory, adjusting MEMEND in the process to protect themselves. I wanted to put them into spare PROM space and use as little RAM as possible to implement the hard disk system. Also their drivers are for the XEBEC S1410 NOT the S1410A and I wanted to use the additional features provided by the latter. One nice thing the S1410A does that the S1410 does not do is write verify; which means it will verify a sector just written transfering only the STATUS without transfering any data. The S1410 requires that you read a sector just written to see if it was written correctly or not. The S1410A will read the sector and give only the status which is much faster! A week or so of blood-sweat-and-tears resulted in software drivers that were ROMable, used the S1410A features, and used only 32 bytes of RAM! It also resulted in the discovery of a few (undocumented) differences between the S1410 and the S1410A controllers and a couple of bugs in WellWritten's software (due to the modifications to allow the use of two (physical as opposed to logical) drives. After some "brainstorming" with Tom (there goes the phone bill again), we worked out the problems and the moment of truth was close at hand. It was time to connect everything up and see if it would actually work!

DISASTER STRIKES! When I applied power, the fuse blew on the JRD Winchester Enclosure/power supply! Every time I applied power the same thing happened! So I finally replaced the fuse with a "26 GAUGE copper" fuse and began connecting test equipment to the power supply. It did not take long to discover that the startup current of two winchesters and a XEBEC controller was somewhere between that of an arc welder and an aluminum processing plant! The JRD enclosure/power supply that I bought from Priority One was NOT up to the task! And I don't care what the advertisements say, she won't do it! The easiest way around the problem was to add a couple of switches so each drive could be turned on or off individually. Which stopped the fuses from blowing; but did little for the bright red glow from around the regulator transistors. After adding about twelve pounds of finned heat sinking pilfered from an old Collins microwave system the pass transistors glowed an acceptable color! And everything ran at a temperature which would not lead to premature failure. But there is no way that I can recommend the JRD enclosure from Priority One Electronics! I do understand that Triple A in Chicago has a similar enclosure/power supply that is up to the job, and there are probably others.

WellWritten's approach to the host adaptor card was somewhat different than what I had originally intended. They used TTL chips to build the host adaptor, while I had planned to use a PIA. Their way is better! Using a PIA requires that one side is used for the SASI control lines while the other side acts as a bi-directional eight bit data bus. Using the PIA means that it must be initialized by the software and reinitialized whenever it is necessary to change the data direction on the data side. Using discrete IC's means that no initialization of any kind is required for the host adaptor and the read-write line on the SS-30 bus is used to change the data direction. So the PIA approach loses in two ways, more software overhead and slower. This may not appear very important, but Winchesters connected to the SS-30 bus and used thru program controlled IO seem quite slow to anyone accustomed to using hard disks! This is because Winchesters normally use DMA controllers which tend to be blindingly fast! But the trade-off is simple, DMA hard disk systems cost lots and lots of $$$$$$. But if you have not been accustomed to using a DMA hard disk system, the difference between floppys and WellWritten's hard disk system is very similar to the difference between tape and floppys! Tom Weaver at

WellWritten Enterprises tells me that I would be a lot happier with the speed of the hard disks if I were using OS-9 instead of Flex. This is because hard disk systems of this type work about ten times faster under OS-9, according to Tom. He is probably correct, but I have been using Flex for years and have a large investment in software that runs under Flex. In a way this saddens me, you see, Ken Kaplan at Microware is one of the nicest people I have come to know thru computer hacking. And I would truly like to be using his product because I happen to like him. Such is life, I guess. I like Richard Don of GIMIX too, but I can't afford anything with the GIMIX name on it!

I may have given you a false impression about WellWritten's software by all the talk of the modifications necessary. It worked well straight out of the box and appears to be completely "bullet-proof". The modifications I made were only necessary because I wanted it to work out of PROM and therefore be as compact as possible; wanted to use the advanced features of the XEBEC A-model controller; and (probably most important) wanted to have the fun of re-doing it! The software consists of the subroutines to allow your operating system to use the Winchesters and a format utility to let you format a new drive (which takes FOREVER) or re-format an old drive (QUICK). I am still looking for a good way to boot directly from the hard disks (and so is Tom, I think) so if anyone knows how to do it please drop me a line!

WellWritten should be commended for providing the source code for their software! That used to be normal, but is now quite rare! If you also have an IBM (or clone) you know that those folks consider the source code to be more valuable than the Queen of England does the "crown jewels". I'll never understand that, because I have yet to see ANY software for the MS-DOS machines that was worth the asking price! We have the PC-6300 because we have one at work and needed to compatibility at home and it is BY FAR the best MS-DOS machine I have seen. But it is sadly lacking when compared to this antique I am using at the moment. I guess it will take another generation or two (of people) to prove wrong the head of a LARGE corporation who recently told me; "You are wasting your time to go into a boardroom for approval of a major computer purchase unless the equipment you are recommending has IBM written on it."

Adding the hard disk drives has pointed out a major shortcoming of the Flex operating system to me. Flex was developed for use with floppy disk systems and is a fine single user single tasking operating system for computers using floppys. When you add hard disk drives, and load them with software a DIR command seems to go on for DAYS. The operating system I use most is UNIX which allows (and encourages) the use of sub-directories. Flex has only one directory which is not good when you have hard disks. This may prove to be another powerful argument for OS-9 which is very UNIX like. A directory with two or three hundred files in it can be most confusing!

If I ramble on much further, 68 Micro will be forced to blue pencil most of what I have said. So I had better end this pretty quickly! As I have described, I have added TWO hard disk drives to my existing 6809 system for well under $1000. All of the software I already owned runs perfectly with the hard disks. WellWritten Enterprises hardware is very good and their software works without any hassle or bugs. It was easy to get the hard disk system running. MS-DOS syatems STINK! And, If you have been thinking about adding a Winchester; give Tom a call. I think you will be glad you did!

In case you were wondering why I said that I think I am in trouble with my wife; the hard disk system has made using this computer so very much more enjoyable that I have been spending too much time playing with the old computer. If she were not upstairs on the PS-6300, I KNOW I would be in serious trouble!

** Editor's Note: Well Mickey, you give me a good chance to get my licks in, again. Those "old hackers" are still around, and a lot of potential new ones also, but it is a new ballgame.

The simple fact is, we (hackers) don't return enough profit (mostly none). We want kits and bare boards at reduced prices. We sometimes use "surplus" - spell more like "drop-out" parts (save a few cents, you know) and then when it (board or whatever we bought naked as a blue-jay) flakes out we either bend the vendors ears to the extent he tearfully regrets ever have gotten into the "bare" board business or we write a nasty letter-to-the-editor and cause the poor guy even more grief. Or, occasionally he runs into a guy like you (and I like to think, me) who does his homework, realizes that the kit and bare board dealer has this problem and appreciates his hanging in there with us. And when we do tie him up, it should be at least, mutually benefiting.

For all the time he spends on the phone explaining the circuit in minute detail to an earnest but "wet behind the ears" aspiring hacker, writing letters to different magazines explaining why his board or what-have-you normally does not fizz up in smoke, scratching his head as he looks over a board received in the mail with a letter demanding an immediate refund (plus postage and something for lost time), noticing that all the foil is lifted from the board, apparently because far too much voltage was applied, or the regulator sit there, crisp on the board, soldered in backwards, etc. Or a demand for an immediate refund because it won't work with a surplus Dig-A-Wumpus lens fogger, that was a real $1.67 bargain (has to be, has over 100 ICs on it) - Hum, wonder what RTL or DTL means, oh well. And so it goes.....

And I can recite a few hundred more reasons why the kits and bare boards disappeared. NO PROFIT!!!!!!!!!!!!

Now for folks like you, me and thousands of others, that has been a tragedy! But the hard cold facts are, too many abused or misused the do-it-yourself suppliers. So they joined the ranks of the "we will do it all, then they can't (we hope) mess it up too bad - tie us up on the phone for hours, etc.". Hence, all ready-rolled lately.

We get calls all the time from "new blood" wanting to know what kits, etc. are available? Fact is, a majority of us who now buy the factory made stuff, actually were once the "hot on kits" types, only the kits and bare boards disappeared. Without kits, this thing would have never gotten to the place where it is now. And probably with nothing but kits, it still would not have gotten from there to here. Sort of a catch-22 thing. But the fact remains that there is a multi-million dollar market out there, all wanting kits. Most would pay "premium" prices just for the educational benefits of constructing a kit. But the bitter taste is still too fresh.

Some day, some smart guy is going to figure it out right, get the necessary capital together, buy out the rights to some kit manufacturers (I know one who wants to sell) computer kits and boards. And be another instant millionaire. The market is certainly there. A recent survey showed that kits and bare boards are the deepest void, that has a ready market, in the whole computer game. But it also is fraught with sink holes, it has to be done right, with a decent margin figured in to cover all the extra cost a kit vendor experiences. Beyond that, it is just another sound business venture, with better than average odds to succeed. And we understand that- education cost!

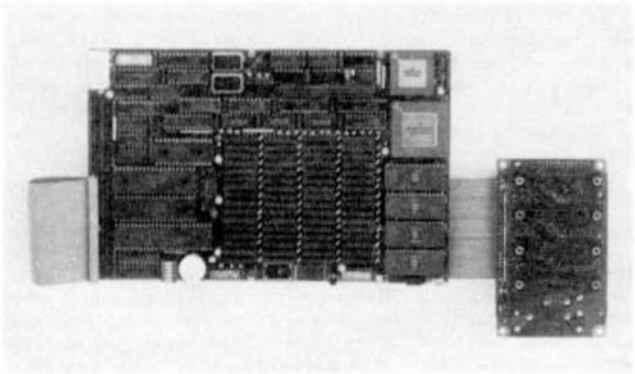
DMW

- - -

MUSTANG - 020

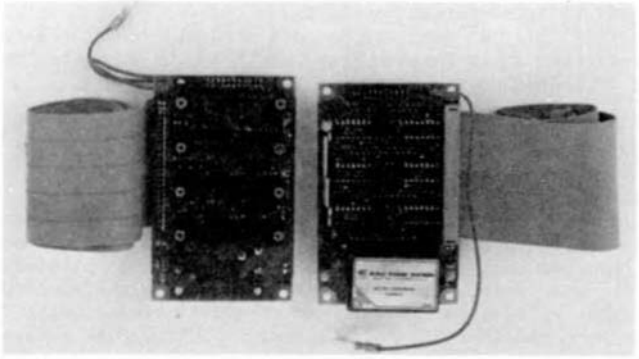
By the time you receive this, our Data-Comp Division should be into the first deliveriee of their Mustang-020. The response was more that I had actually expected, yet I had felt that it would be strong.

When you consider that the MUSTANG-020 is a full 12.5 megahertz machine, with over 2,000,000 bytes of RAM and comes wired/teated, even the 4 port cable and connector plate, with 4 DB25 pre-wired connectors and a 12 volt to 12 volt de converter. Also included is the Motorola 020 Bug, which normally sells for around $500.00 alone. It is hard to comprehend that all that P O W E R and S P E E D can be completely contained in such a small enclosure.



We had the whole thing, enclosure, cables, floppy and hard disk, configured and going around the first of December.

You will also note that we are changing our recommendation as concerns hard disk controllers. At first we thought that both the OMTI 20C and 5000 series as well as the Xebec 1410 and 1410A controllers would do. However, as the prototyping continued we found that the 5000 series OMTI would not do at all and the OMTI 20-C1 was the only OMTI that we could recommend (because of pre-comp). The OMTI 20-C will probably work if your hard disk does not require pre-comp, but because of the varied specifications for several different brands of hard disk available and driver changs, we decided to not include it in our regular offering. The Xebec 1410/1410A both work well and although not quite as fast as the OMTI, will do for any reasonable application (our prototype has a five year old Xebec 1410 installed). However, if you need that extra speed (about 12-22% in some modes) then the OMTI 20-C1 is worth the extra $200.00. And we do furnish the specs for most of the more popular hard disk drives so that you can configure to the OMTI 20-C1. If you desire to order the OMTI 20-C1 be prepared for the additional cost of $200. As a result, unless you order the OMTI specifically, we will install the Xebec 1410 series controller. The darn thing is aoooooooo fast now that the little difference does not seem to make much big difference, but if you need it, we can install it (OMTI 20-C1). My personal feeling on the entire matter is to invest in the 68881 co-processor, and use the Xebec controller. Makes the cost of the 68881 a little less



biting. And the speed lost with the disk drive controller is recovered and plus, in the gained speed and efficiency of program execution. Which is actually what the system does most of the time, i.e. - time spent in program execution as opposed to disk loads and saves. Of course, you can always install both and be on the very top!

On our prototype we set up a RAM disk (/dd - default drive) consisting of 750K of RAM. This still leaves us with over a million bytes of free RAM! When we login it loads the entire CMDS directory into /dd RAM Disk and also all the LIB, DEFS, SYS and other directories needed for BASIC09, C, PASCAL and all other normal operations. The entire load from hard disk to RAM disk is about 42 seconds. That consists of about 680,000 bytes transfered. We then do everything from RAM disk, and occasionally update load (via date reference; i.e. if a later date and time than when loaded, then copy (update) to hard disk). The normal update operation only consumes about 3 to 4 seconds. Because of the tremendous speed of the MUSTANG-020 it is a simple, safe and above all fast way to utilize this kind of computer.

Also we are finding that most all our HLL programs can be ported directly to the MUSTANG-020, recompiled and run. 6809 and 68020 BASIC09 is very close. In most cases no changes at all needed. And the MUSTANG-020 uses the same disk format as OS-9 6809. No porting problems whatsoever.

Another plus is that this version of OS-9's C compiler is very, very close to UNIX System V 'C'. Most System V software can be ported over and run with little if any modification (per Microware documentation). This feature we have not tested personally, but others more knowledgeable than I have said likewise, so I repeat it here now. That being the case, it is to be assumed that it would work the other way also. If any of you know for

sure, let me know and I will pass it along.

Another interesting and powerful feature is that one floppy disk is really all that is needed with the MUSTANG-020. At first, because of the slow manner of doing say a BACKUP with our 6809 OS-9 level III system, and one disk drive, being able to only use 64K, 96 tpi (80 track) OS DD, we felt that maybe two drives would be needed. As it works out, one is sufficient. So far about the only thing we use the floppy drive for is porting into and out of the system. BACKUP on the 68020 is assigned 750K of buffer RAM and swallows the source disk in one sweep. Then just insert the destination disk and within less than a couple of minutes or so 600,000 plus bytes are backed up to the new disk. Takes a little longer to a floppy than to a hard disk or RAM disk. We ship all floppies (80 track, DS DD) running at 6 ms. however, I recommend that you retune them (fast and easy with the furnished source makefile and MAKE command) to 3 ms., as all the floppies we ship are capable of 3 ms. access speeds.

Also you might bear in mind that additional I/O options are to be made available for this system. First an additional 8 port expansion unit will be offered (total 12 users or devices (printers, modems, plotters,etc.) and later for UniFLEX intelligent I/O expansion units. There is no reason that this system should not be capable of supporting 32 or more users. The future looks bright!

In addition we hope to be offering an expansion, plug-on RAM card, about another 2 meg or so. Also don't be surprised to see some very fine CAD software from Microware and Hitachi also offered in the not too distant future. Pretty exciting for all of us here. And one nice thing that Motorola has said that makes me feel better about this whole project is that they expect the 68020 to hold a strong position (support and all that other good stuff) well into the 1990's. As you might remember I often expressed the feeling that I personally felt that they had not supported some of the other devices as well as they might have.

For you speed freaks, or those who need even more speed, then I would suggest that you consider UniFLEX. It is faster in most instances. While much of OS-9 is compiled from 'C' source, I suspect that most all of UniFLEX is assembler. Also our UniFLEX 68020 system is optimized for the 68020, not just the system but the 'C' compiler and other TSC MHLs as well. Although UniFLEX is higher in cost by about $100 dollars, it has a lot going for it. In the months to come we will be doing a comparative review of several systems, as well as a STAR-DOS (FLEX) compatibility 68XXX system (watch this one - a possible sleeper!).

You might want to bear in mind that you can get closer to the internals of the machine with OS-9 than UniFLEX, as the OS-9 system comes with source or documentation for practically everything needed to attach about any device you might desire to the MUSTANG-020 under OS-9 (far more latitude in being able to configure this system to your hardware applications than the 6809 versions). As for UniFLEX, while very complete in driver selection for a number of devices, you do not get source or any other information to assist in developing drivers, device descriptors, etc. That has been a long time complaint of mine; TSC has not, despite promising on several occasions some years back, to release a version of UniFLEX. I and others were told by TSC that it was done but only the documentation was holding things up. In a more recent conversation with someone from TSC I was given the impression that the later versions were in fact. But still no configuration documentation. I wonder why they put the configuration section in the UniFLEX manual, but never saw fit to deliver, as promised at several SWTPC dealer meetings?

Are we really all that stupid that we (all of us!) would screw up if we attempted to configure UniFLEX? I do know that thousands have been configuring OS-9 for several years (from day one actually) and they seem to do pretty well, and so has Microware. Even the many thousands of CoCo OS-9 user manage! I wonder what the lesson is there?

In that conversation I was told that TSC was willing to write any drivers or descriptors that you might need. For a price. However, based on my experience with the marketplace (as a consultant I have been responsible for several multi-tasking, multi-user system decision by some rather large companies, both domestic and abroad) and most of them have engineers more than capable of doing normal configuration projects. Fact is some balk at having to pay a vendor to do additional functions that could be accomplished inhouse. After all, that is what they hire and maintain an engineering staff for.

Right now I have a client who is a 150 unit site-license prospect. They have an excellent engineering staff. The folks in engineering are reluctant to recommend a system that must have outside support for the more mundane projects, also it cuts into their justification for being there in the first place. But system power and speed are prime considerations. Both systems have their merits, and each can do a bang-up job of milking all the power out of the 68020 that will be reasonably expected. I guess it is all in how they view the overall picture. As of this date no decision has been made, but it is certainly a nice little contract.

Another consideration when deciding on your MUSTANG-020 system is the Floating Point Co-processor, the 68881. I strongly suggest that you decide prior to ordering the system. The reason being that the soft/firmware has pointers to the 68881 co-processor. As the system comes burned in and ready to run, the soft/firmware must know about the 68881 being onboard (to handle the 68881 code). It can be installed later, but you are in for some programming effort. So consider the above carefully, as it can save you time and money to order it in the initial order.

The 68881 co-processor is a complete processor in it's own right. Fact is, it is probably more complex than the 68020. As it was designed especially for the 68020 (but will work with the 68010, 68012 also) it makes a big difference when it comes to crunching numbers. If you are primarily processing text then I would recommend saving the difference. The 68020 is no slouch, even without the 68881!

In the coming months I will be publishing additional information concerning the MUSTANG-020 and other 68XXX systems. Also will be telling you about our experiences with the system. We are learning new and interesting things every day. Our prototyping has been an eye opener, to say the least.

There is one thing that I can tell you for certain. **The MUSTANG-020 is the most powerful system we have ever had in our office or lab (we once had a 360). It has more power and speed than most all other systems selling in the $25,000.00 to $150,000.00 dollar range!** I know that is a big claim, but if you don't believe it, look again at the benchmarks. Go out and price say a VAX 11-780, with hard disk and 2,000,000 bytes of RAM. Please note the benchmarks listed below.

Program USR as written in the language "C".

Program should be executed (6) times with the following types of variables:

short.
register short.
integer.
register integer.
long.
register long.

```
main ( )
{ (TYPE) i,j,k;
  for (i=0; i<1000; i++)
     for (j=0; j<1000; j++)
        k = i + j + 1983;
}
```

UNICOM/SAN DIEGO BENCHMARK RESULTS

Program USR as written in the language "SMPL".

Register assignment is assumed to be a compiler responsibility. Program should be executed (3) times with the following types of variables:

word,
pointer,
long.

```
type tp = long;
procedure main
  variable tp i, j, k;
  for i=1 step i+1 while i <= 1000
    for j=1 step j+1 while j <= 1000
      k = i + j + 1983
    end { for j }
  end { for i }
  return
end { main }
end { procedure }
```

Program USR benchmark results for various architecture machines.

Machine (Time in seconds)	CPU Mhz	USR Short	USR Short Register	USR Integer	USR Integer Register	USR Long	USR Long Reg
Elite Consultant with cache							
*NS32016	10	3.71	3.71	3.71	3.71	3.71	3.71
*NS32016 with MMU	10	4.64	4.64	4.64	4.64	4.64	4.64
**NS32032	10	3.10	3.10	3.10	3.10	3.10	3.10
**NS32032 with MMU	10	3.20	3.20	3.20	3.20	3.20	3.20
DEC VAX 11/780		8.8	8.6	6.7	4.7	6.7	4.7
VAX 11/730		37.4	37.5	23.9	14.4	24.1	14.4
PDP 11/44		11.2	5.7	11.2	NA	21.8	21.8
LSI 11/23		34.5	14.9	34.6	14.9	72.2	72.2
MC68000's							
Pacific	10	11.5	10.7	20.7	9.7	20.8	9.7
Sun	8	12.0	7.2	13.6	NA	13.6	6.6
Altos	8	13.9	13.9	13.9	13.9	17.8	17.8
Codata	8	17.1	10.5	19.3	NA	19.3	10.5
Cornix	8	19.8	11.6	22.5	10.1	22.6	10.1
Fortune 32:16	8	21.7	12.8	25.0	12.4	25.0	12.3
Apple Unisoft	5	22.6	13.7	25.9	12.1	25.9	12.1
Apple Xenix	5	37.9	23.1	43.4	21.7	43.1	21.6
Wicat NS150	8	24.8	14.4	28.1	13.1	28.1	13.1
Ch Rivers OS	8	20.2	15.8	31.7	NA	31.7	15.8
IBM PC Xenix	8	30.3	17.6	34.2	17.3	35.1	16.9
TRS80-Xenix		25.2	14.9	20.3	14.0	28.4	14.2
Pixel 100/AP	8	18.6	11.6	21.5	9.6	22.9	9.6
Z8000's							
Zilog	6	14.7	7.3	14.7	NA	25.7	13.3
Plexus	5	15.2	7.0	15.4	7.0	27.5	27.6
8086							
Altos	10	13.7	7.2	13.7	7.2	27.8	27.7
Other							
NS32016/Mesa	4	49.9	45.0	56.7	23.3	57.2	27.2
PE 3210		16.7	6.7	15.9	NA	15.9	6.7
Perq (ICL)		NA	NA	10.2	NA	NA	NA
Gould Concept 32/87		1.9	1.9	1.7	1.2	1.7	1.1
*MEASURED							
**EXTRAPOLATED							
Mustang 020-UniFLEX VM	16.5	2.97	1.83	2.85	1.71	2.84	1.72
OS9/68K	16.5	4.22	2.57	4.45	1.59	4.45	1.60

As a last bit of information this time, I might suggest that you opt for the 20 megabyte size hard disk when ordering the MUSTANG-020. Because of our quantity purchase of hard disk drives, the difference between the 10 and 20 megabyte drive is only $200 to you (current prices, subject to change due to an unstable disk market) and the difference is well worth the small additional expense.

We hope to be able to ship installed and tested systems within 15 days of receipt of firm orders. Boards alone will take a little less time. The time difference is that we do a tight-margin burn in for boards alone. For complete systems the margin is tighter due to floppy and hard disk burn-ins. However, the demand is ahead of the supply, at this time. In the next few months we should get pretty well caught up. But worst case, as it appears now, is 30 day turn around. Hope Murphy isn't reading this!

For those of you with a Q--- something or another 68XXX SBC, and wishing to take advantage of our trade-in offer ($400.00 if it still works and has all the software and documentation you received with the system), give us a call. However, so that you might know now, basically this is the way that the trade-in thing will work. If you want to ship us your trade-in prior to our shipping you the MUSTANG-020 we will test it out and you will only need to enclose the difference with your order. Otherwise we will ship at the advertised rate and issue a rebate check for the trade-in value as soon as we receive and test your trade-in. Also we will be having a few of these (trade-ins) available as we are already receiving trade-ins. These we will be selling, without any warranty, at a price close to our break-even cost. Call or write for quotes. We can even install them in our enclosures with our disk drives. But remember we WILL NOT pass along any warranty, for any part of the system using these used SBCs.

Which brings up one last point. Our enclosures are ideal for any SBC 68XXX or otherwise. The power supply is the switching type and top grade. The enclosure can hold two half high drives and a mounting bracket is included. Most S8Ca can be attached to the bottom of the system, top of the bracket or to one of the drives (as we do). The enclosures are ac/rf by-passed, with a lighted power switch and another front panel switch for reset. Several cut-outs on the back to hold connectors. etc. The enclosure is a fan ducted system, with bottom and front side ventilation intakes. All wiring is included for standard power connections to 4 devices (5 vdc and +12 vdc).

DHW

- - -

FLEX - 09 KERMIT

By: Jur van der Burg
Nettelhorst 56
2402 LS Alphen aan den Rijn
The Netherlands

Language: C (Compiled with Introl (c) compiler)
Version: 2.3

Date: November 1985

KERMIT for FLEX is derived from the UNIX version.
It is enhanced in several ways, such as data logging, server mode etc.

It should run on about any version of the FLEX-09 (tm) operating system. Hardware dependent things are kept in the files FLK.H and FLIO.C .

Command summary:

CONNECT

Format: CONNECT

The CONNECT command will allow you to make a connection to the remote system over the line that was specified by the SET LINE command. If a log file was opened (SET LOG) then the data will be buffered in memory. If this becomes full, it will be written to disk. Handshaking is provided (see SET HANDSHAKE).

While connected, several sub-commands are possible.
These are the escape character arguments:

C	Close connection, "escape" to command mode
S	Show status of connection
Q	Quit logging
R	Reaume logging
H	Show availability
B	Send 'BREAK' signal
F	Execute FLEX command
0	Send null
?	Show escape character arguments
escape char	Send escape character itself
Other	Rings the bell

SEND

Format: SEND file [,file [,file...]]

The SEND command will allow you to send a file(s) to the other Kermit. The drive number specification will be stripped off the filespec.
The current transfer can be aborted with control-C.

RECEIVE

Format: RECEIVE

The RECEIVE command will allow you to receive a file(s) from the other Kermit. The filenames will be specified by the sending (remote) Kermit.
The current transfer can be aborted with control-C.

GET

Format: GET file [,file [,file...]]

The GET command will allow you to get a file(s) from a remote server by specifying their names.
The current transfer can be aborted with control-C.

SERVER

Format: SERVER

This command will cause Kermit to enter server mode. The other Kermit can then issue server commands to send and receive files without having to give SEND or RECEIVE commands to Kermit-09. Type a Control-C to quit the server mode. After a timeout the server will quit.

FINISH

Format: FINISH

This command will case Kermit-09 to tell the other Kermit (Which should be in server mode) to exit from Kermit. After receiving acknowledgement that this is being done, Kermit will prompt for another command.

BYE

Format: BYE

This command will cause Kermit-09 to tell the other Kermit (which should be in server mode) to exit from Kermit and, if applicable, terminate its job (or process, etc.). After receiving acknowledgement that this is being done, Kermit will exit to FLEX.

HELP

Format: HELP

This command will give a short display of the available commands.

LOAD

Format: LOAD file

This command will send a textfile to the current line.
It can be used to send a file of parameters to a smart modem, or to send bare text to the remote system.

FLEX

Format: FLEX [command]

This command allows you to execute a FLEX command from within Kermit. When no command is given, it will be prompted for. Be careful only to use FLEX commands which run in the utility command space!

STATUS

Format: STATUS

This command shows the current status of Kermit-09. This includes the number of characters that have been send and received from the remote Kermit. When a real-time clock is available, an estimate of the effective baud rate of the transfer is included.

EXIT

Format: EXIT

This command will exit Kermit-09. If a log file was used with CONNECT, and the buffer contains still data, then the buffer will be written to disk before terminating Kermit-09.

The SET command is used to set various parameters in Kermit.

SET

Format: SET command

LINE

Format: SET LINE address

This command will the the address of the interface you are using for the transfer. A check will be made to see if the interface is available.

BAUD

Format: SET BAUD speed

This command will set the desired baudrate for the communication. In simple FLEX systems the only baudrates allowed are 300 and 1200. This is switched by the divider of the ACIA interface.

CONFIGURATION

Format: SET CONFIGURATION number

This command will set the configuration to use for the communication. The following values are possible:

0 - 7 bits, even parity, 2 stop bits
1 - 7 bits, odd parity, 2 stop bits
2 - 7 bits, even parity, 1 stop bit
3 - 7 bits, odd parity, 1 stop bit
4 - 8 bits, no parity, 2 stop bits
5 - 8 bits, no parity, 1 stop bit
6 - 8 bits, even parity, 1 stop bit
7 - 8 bits, odd parity, 1 stop bit

ESCAPE

Format: SET ESCAPE character

This command allows you to set the ESCAPE character for the CONNECT processing.

DEBUG

Format: SET DEBUG on/off

This command will make the transfer more verbose. It will show the received and transmitted packets, and various other things of interest.

TIMEOUT

Format: SET TIMEOUT value

This command will set the number of seconds before Kermit-09 will time out to attempt to receive a message. This timeout is used to handle transmission errors which totally lose a message. The default value is five seconds.

IMAGEMODE

Format: SET IMAGEMODE on/off

This command will set the image mode transfer flag. When it is set, the data will be transferred as it is. This is useful for transfer between FLEX systems. When clear, space compression will take place, as well the expansion of tabs. Carriage returns will not be processed, while a linefeed will be converted to a carriage return.

DUPLEX

Format: SET DUPLEX full/half

This command will control the CONNECT command, in that the locally typed characters will be echoed to the local system when duplex is set to HALF.

LOG

Format: SET LOG file

This command will specify a file to capture the data received from the remote system within the CONNECT command.
The file will be closed if the filespec is a dash ("-").

HANDSHAKE

Format: SET HANDSHAKE start stop

This command will set the handshake characters used for controlling data within the CONNECT command. The defaults are XON and XOFF. When the capture buffer becomes full, the stop character is sent to the remote system. If that system stops sending data, the buffer will be written to disk. After that the start character will be sent.

The SHOW command will show all parameters set with the SET command, with one extension: SHOW ALL.

Format: SHOW command

- - -

Editor's Note: Due to the size of KERMIT, the source is available, for Ictrol C, on both 8 and 5 inch disks. The entire set of programs and other data consist of about 730 FLEX sectors. Therefore, the price of the 8 inch disk is as advertised in the "Reader Service Ad", this issue. For a 2 disk - 5 inch set - the price is increased from $12.95 to $19.95, to cover additional cost.

DMW

- - -

SWTPC DMAF2
Modified To Support
5" DRIVES

Egbert Jan van den Buaache
Raam 50-A
2611 LV DELFT
HOLLAND

With the appropiate modificationa to the board it's very well poaaible to combine 5" an 8" drives to work via the DMAF2 controller. No apecial treatment of the drivea is neceaaary, but if the 5" drivea have a ateprate which is half aa faat aa the 8" drives it makes things pretty easy All aignals are the aame only the data transfer rate is different. The 5" drivea have a transfer rate of 125 Kbit/aec where the 8" drives have 250 Kbit/aec (single denaity). The aame applies to double denaity, where they do 250 Kbit/aec (5") and 500 Kbit/aec (8"). To connect 5" drivea to the DMAF2 controller only a twisted cable is required which connects the 34 pin 5" connector to the 50 pin 8" connector. The DMAF2 board needs several patches.

1. The WD1791 floppy controller chip must be supplied with 1 MHz clock instead of 2 MHz.
2. The reference frequency for the PLL must be changed.
3. The time constant of the input circuit must be changed.
4. FLEX must be adapted.
5. UniFLEX must be adapted.

1. and 2. are easily aolved by adding two additional "divide by two or not" circuits. In fact these are the aame aa already in uae for the Single/Double denaity selection. See fig. 2, where the controlling signal is not DDEN(-) but 8"(-). Thia requires one 74S112 and two 74LS00. To fix the PLL cut the trace near pin 6 of IC15 and connect pin 6 to pin 1 of the 74S112 connect pin 6 of the added 74LS00 to the trace previously going to pin 6 of IC15. The other controlled divide by two ia placed in the clock going to the WD1791. Cut the trace from IC17 pin 6 to IC20 pin 24 and insert here the second "divide by two or not".

The 8"(-) signal is derived from the drive select latch where a two input OR is looking for a "1" on pin 6 or 15 of IC25. Connect pin 1 and 2 of the spare gate in IC34 to pin 6 and 15 of IC29. The output of this gate is the 8"(-) signal.

As aoon aa drive 2 or 3 is aelected one of the 5" drives is in use and is the controller aet for 1 MHz operation. To aatisfy the third item we need aomething more. SWTPc uses a tranaiator awitch to decreaae the time constant of the input filter, but now we have to awitch between three valuea. I replaced the original circuit by that of figure 1. The adjuatment is even more easy; one variable reaistor for each time constant.

Moat problems are aolved now. But... the circuit uaed for write precomp is not working correctly at 1 MHz! Careful atudy of the WD1791 data aheet learns that there is a ahift in the phaae relation of clock and data. As my TANDON 848 drivea refuaed to work with precomp anyway I bypaaaed the whole circuit. Cut IC20 pin 31 to IC18 pin 12 and IC6 pin 13 to IC12 pin 17 and connect IC20 pin 31 to IC20 pin 17.

The aoftware (FLEX).

The drivers in FLEX are only suited for one type of drive. Main problem is a wrong aector count before awitching to the other aide of the diak or to the next track. I aolved this by building aome intelligence in the driver. As aoon as drive 2 or 3 is current the aector counts are adjuated. See liating of driver overlay. All the reat is taken care of by the hardware.

To be able to create 5" diaks you need a NEWDISK program. I uaed the atandard TSC NEWDISK which I had on my diak (8" DMA veraion), the WESTERN DIGITAL manual and DYNAMITE+ to create another NEWDISK program for 5" DMA. Eaaential to know is the byte count per track which is 3100 bytes for a aingle denaity track and 6200 bytes for a double denaity track.

In UniFLEX the awapping on 5" floppiea is alower than on 8" drivea. Aa we have 5 or 10 aectora (512 bytea) on a 5" UniFLEX diak we cannot write 8 aectora after each other without awitching aidea, aa is done on 8" diaka. (8 timea 512 is 4K which is the minimum memory bank aize). The aolution is to awap block by block... It worka but is alow.

I do not want to go in detaila about UniFLEX, I'am atill waiting for reply from TSC regarding to what extend I can publiah modificationa to copyrighted atuff.

Concluaion.

Thia writing is not to get everybody digging in their DMAF2 boarda but it again atreaaes the extreme flexibility of our SS50 hardware which is atill around when many otherа diaappeared from the acene. Alao it givea an idea to what level my hardware is upgraded. I hope to take you into UniFLEX next time.

Don, thank you for printing thia in your fine magazine. I hope to aend you more from the Dutch CS/SWTPc User Group in the near future.

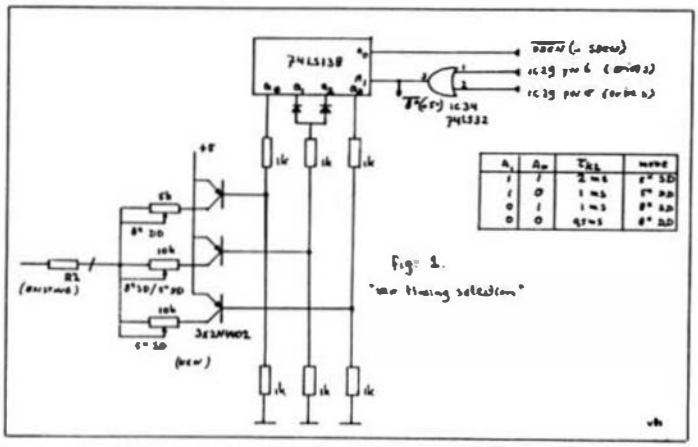
- - -

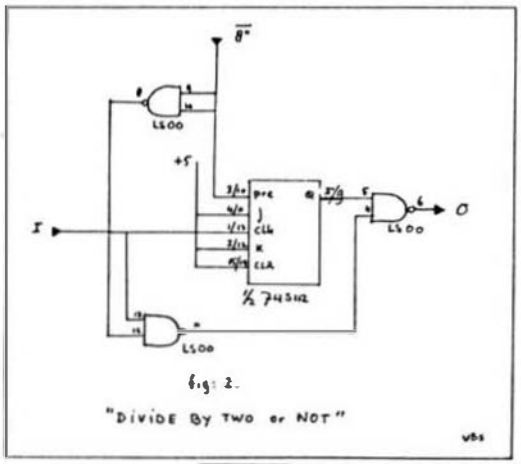
Editor's Note: Thanks Egbert for thia information. everytime I publiah an article of thia nature I get heapa of good commenta.

All ahould remember that the S50 bua is far from dead. Maybe not aa active aa once waa, but atill a very active group. The exiating hardware, for the moat part is top quality, and atill auperior to moat of the atuff being aold by the 'other aide'.

If other readera who have updated their equipment keep thia aort of material coming, we have a very bright future for the hardware typea alao. How 'bout it gang?

DMW






```
newdisk3.sre
dsm formatter for 3" drives


        * standard pre-named label equates

CCBC    srsdat  equ     $ccbc
CDB3    nerec   equ     $cdb3
CD15    setchr  equ     $cd15
CD18    selchr  equ     $cd18
CD1B    inbuff  equ     $cd1b
CD1C    cntrno  equ     $cd1c
CD24    nerif   equ     $cd24
CD2D    setfit  equ     $cd2d
CD39    nuldoo  equ     $cd39
CD3C    nothre  equ     $cd3c
CD42    prther  equ     $cd42
CD48    imnec   equ     $cd48
D4B6    fes     equ     $d4b6
D4B3    fatcls  equ     $d4b3
DE01    derite  equ     $de03     system drive ss
DE09    drest   equ     $de09     system restore
DE1B    dseek   equ     $de1b     system seek

        * ascii code equates

0008    nul     equ     $08
0004    eot     equ     $04
000A    lf      equ     $0a
000D    cr      equ     $0d

        * external label equates

0100    work    equ     $0100
0101    first   equ     $0101
01DC    sekend  equ     eork+this
1938    dskend  equ     this+2*work
C07F    stack   equ     $c07f
F000    ch0cdr  equ     $f000
F002    ch0brt  equ     $f002
F010    ch0cha  equ     $f010
F014    prtcha  equ     $f014
F020    fdstat  equ     $f020
F023    fddata  equ     $f023
F024    drvsel  equ     $f024
F040    autodr  equ     $f040
F012    lrn     equ     $f012     lrn translater routine
FFFF    ffff    equ     $ffff

        * disk parameters

0028    sectrk  equ     40
0008    secsiz  equ     18
0014    secssl  equ     20
0012    dsecs0  equ     18
0024    dsecs1  equ     36
```

SOUTH EAST MEDIA

New Software Additions

TEXT EDITORS

CEDRIC – A screen oriented TEXT EDITOR with availability of "MENU" aid. Macro definitions, configurable "permanent definable MACROS" – all standard features and the fastest "global" functions in the west. A simple, automatic terminal config program makes this a real "no hassel" product. Only 6K in size, leaving the average system over 165 sectors for text buffer – appx. 14,000 plus of free memory! Extra fine for programming as well as text.

Regular $129.95

*** SPECIAL INTRODUCTION OFFER *** **FLEX $69.95**

- - - - - - - - - - -

PAT – A full feature screen oriented TEXT EDITOR with all the best of "PIE (tm)". For those who swore by and loved only PIE, this is for you! All PIE features and much more! Too many features to list. And if you don't like these, change or add your own. PL-9 source furnished, "C" source available soon. Easily configured to your CRT, with special config section.

Regular FLEX $129.50

*** SPECIAL INTRODUCTION OFFER *** **FLEX $79.95**

SPECIAL PAT/JUST COMBO

PAT & JUST (w/source) **FLEX $99.95**

Note: JUST in "C" source available for OS-9

☞ See JUST advertising – S.E. MEDIA Catalog – this issue

PROGRAMMERS & USERS TOOLS

SOLVE – OS-9 Levels I and II only. A Symbolic Object/Logic Verification & Examine debugger. Including inline debugging, disassemble and assemble. **SOLVE IS THE MOST COMPLETE DEBUGGER we have seen for the 6809 OS-9 series!** SOLVE does it all! With a rich selection of monitor, assembler, disassembler, environmental, execution and other miscellaneous commands, **SOLVE is the MOST POWERFUL** tool-kit item you can own! Yet, **SOLVE is simple to use!** With complete documentation, a snap! Everyone who has ordered this package has raved! See review – **68 Micro Journal – December 1985.** No "blind" debugging here, full screen displays, rich and complete in information presented. Since review in 68 Micro Journal, this is our fastest mover!

Levels I & II only – OS-9 Regular $149.95

*** SPECIAL INTRODUCTION OFFER * $69.95**

☞ NOTE: Please note the special discounts (limited time) of assorted software in the S.E. MEDIA catalog in this and other issues.

Also please note the new policy on documentation of some S.E. MEDIA owned or licensed software products offered in their catalog, and repeated below.

Most all prgrms have the documentation in text disk file format. If you can print it out on your printer, the price is as shown in the catalog for the software. If you want us to print it out, please add $25.00. This is our average reproduction, short run cost. On most all items, the savings is well worth your doing the printing. However, this is only done to help save you hard earned dollars. All other software has vendor furnished documentation included in the price. Another – "your choice" **S.E. MEDIA feature!**

K-BASIC updates are now available. If you purchased K-BASIC prior to July 1, 1985 and wish to have your K-BASIC updated, please send $35 enclosed with your master disk to Southeast Media.

K-BASIC under OS-9 and FLEX will now compile TSC BASIC, XBASIC, and XPC Source Code Files

K-BASIC now makes the multitude of TSC BASIC Software available for use under OS-9. Transfer your favorite BASIC Programs to OS-9, compile them, Assemble them, and BINGO -- usable, multi-precision, familiar Software is running under your favorite Operating System!

SAVE $100.00

K-BASIC (OS-9 or FLEX), including the Assembler
!!! Special !!! ~~$199.00~~ $ 99.49

SCULPTOR

Microprocessor Developments Ltd.'s Commercial Application Generator Program provides a **FAST** Commercial Application Development tool unavailable to the OS-9 and UniFLEX User before. Develop any Commercial Application in 20% of the normal required time; gain easy updating or customizing. PLUS, the Application can also be run on MS-DOS and Unix machines! **Sculptor** handles input validation, complex calculations, and exception conditions as well as the normal collecting, displaying, reporting, and updating information in an orderly fashion. Key fields to 160 bytes; unlimited record size; file size should be held to 17 million records. Utilizes ISAM File Structure and B-tree Key files for rapid access. Input and Output communication with other programs and files plus a library of ISAM routines for use with C Programs. Run-time included w/ the Development package; a compiled Application only needs a Run-time License. Additional charge for Networked Units. Prices for Development Package/Run-time. Discounts available for purchases of 5 or more Run-time Packages.

OS-9 / UniFLEX --			68000 UniFLEX --			MS DOS --		
IBM PC Zenix --	$995.00/$175.00		Altos Zenix --	$1595.00/$265.00		--	$595.00/$115.00	
MS DOS Network --	*	**	UNIX --	*	**	PC DOS --	*	**

* Full Developement Package ** Run Time Package Only Full OEM and Dealer Discounts Available!

!!! Special Buy Out !!! **SPECIAL** - Limited Quantity !!! Special Buy Out !!!

6809 FLEX SOFTWARE		
TSC Flex Utilities	was $75.00	Now only $50.00
TSC Basic	was $75.00	Now only $35.00
TSC Text Processor	was $75.00	Now only $50.00
TSC Flex Precompiler	was $50.00	Now only $35.00
TSC Flex Basic	was $75.00	Now only $50.00
TSC Flex Diagnostic	was $75.00	Now only $50.00
TSC Text Processor	was $75.00	Now only $50.00
TSC Assembler	was $50.00	Now only $35.00
TSC Precompiler	was $50.00	Now only $35.00
TSC Editor	was $50.00	Now only $35.00
TSC Utilities	was $75.00	Now only $50.00
DISKS ONLY		
TSC Extended Precompiler	was $50.00	Now only $35.00
TSC Basic Flex	was $75.00	Now only $30.00
TSC Diagnostics	was $75.00	Now only $50.00
TSC Utilities	was $75.00	Now only $50.00

MANUALS ONLY		
TSC Basic Precompiler	was $25.00	Now only $20.00
TSC Text Editor	was $25.00	Now only $20.00
TSC Debug	was $25.00	Now only $20.00
TSC Mnemonic Assembler	was $25.00	Now only $20.00
COMPLETE 6800 SOFTWARE		
TSC Text Processor	was $50.00	Now only $35.00
TSC Precompiler	was $50.00	Now only $35.00
TSC Diagnostics	was $75.00	Now only $50.00
DISKS ONLY - 6800 Software		
TSC Editor	was $50.00	Now only $35.00
TSC Utilities	was $100.00	Now only $70.00
TSC Assemblers	was $50.00	Now only $35.00
MANUALS ONLY - 6800 Software		
TSC Diagnostics	was $25.00	Now only $20.00
TSC Debug	was $25.00	Now only $20.00
TSC Text Processor	was $25.00	Now only $20.00

!!! Please Specify Your Operating System & Disk Size !!!

PROGRAMMING

ASSEMBLERS

ASTRUK09 from **Southeast Media** -- A "Structured Assembler for the 6809" which requires the **TSC Macro Assembler**. F, CCF - **$99.95**

Macro Assembler for **TSC** -- The FLEX **STANDARD** Assembler.
Special -- CCF $35.00; F $50.00

OSM Extended 6809 Macro Assembler from Lloyd I/O. -- Provides local labels, Motorola S-records, and Intel Hex records; XREF. Generate OS-9 Memory modules under FLEX. FLEX, CCF, OS-9 $99.00

Relocating Assembler w/Linking Loader from **TSC**. -- Use with many of the C and Pascal Compilers. F,CCF $150.00

MACE, by Graham Trott from **Windrush Micro Systems** -- Co-Resident Editor and Assembler; fast interactive A.L. Programming for small to medium-sized Programs. F,CCF - $75.00

XMACE -- MACE w/ Cross Assembler for 6800/1/2/3/8 F,CCF - $98.00

TRUE CROSS ASSEMBLERS from **Computer Systems Consultants** -- Supports 1802/5, Z-80, 6800/1/2/3/8/11/HC11, 6804, 6805/HC05/146805, 6809/00/01, 6502 family, 8080/5, 8020/1/2/35/C35/39/40/48/C48/49/C49/50/8748/49, 8031/51/8751, and 68000 Systems. Assembler and Listing formats same as target CPU's format. Produces machine independent Motorola S-Text.
FLEX, CCF, OS-9, UniFLEX each - $50.00
any 3 - $100.00
the complete set w/ C Source (except the 68000 Source) - $200.00

XASM Cross Assemblers for FLEX from **Compusense Ltd.** -- This set of 6800/1/2/3/5/8, 6301, 6502, 8080/5, and Z80 Cross Assemblers uses the familiar **TSC Macro Assembler** Command Line and Source Code format, Assembler options, etc., in providing code for the target CPU's. Complete set, FLEX only - $150.00

CRASAD from **Lloyd I/O** -- 8-Bit Macro Cross Assembler with same features as OSM; cross-assemble to 6800/1/2/3/4/5/8/9/11, 6502, 1802, 8048 Sers, 80/85, Z-8, Z-80, TMS-7000 sers. Supports the target chip's standard mnemonics and addressing modes.
FLEX, CCF, OS-9 Full package -- $399.00

CRASM 16.32 from **Lloyd I/O** -- Cross Assembler for the **68000**.
FLEX, CCF, OS-9 $249.00

DISASSEMBLERS

SUPER SLEUTH from **Computer Systems Consultants** -- Interactive Disassembler; extremely POWERFUL! Disk File Binary/ASCII Examine/Change, Absolute or FULL Disassembly. XREF Generator, Label "Name Changer", and Files of "Standard Label Names" for different Operating Systems

Color Computer	SS-50 Bus (all w/ A.L. Source)
CCO (32K Req'd) Obj. Only $49.00	F, $99.00
CCF, Obj. Only $50.00	U, $100.00
CCF, w/Source $99.00	O, $101.00
CCO, Obj. Only $50.00.	

DYNAMITE + from **Computer Systems Center** -- Excellent standard "Batch Mode" Disassembler. Includes XREF Generator and "Standard Label" Files. Special OS-9 options w/ OS-9 Version.

CCF, Obj. Only $100.00	CCO, Obj. Only $ 59.95
F, " " $100.00	O, " " $150.00
	U, " " $300.00

PROGRAMMING LANGUAGES

PL/9 from **Windrush Micro Systems** -- By Graham Trott. A combination Editor/Compiler/Debugger. Direct source-to-object compilation delivering fast, compact, re-entrant, ROM-able, PIC. 8 & 16-bit integers & 6-digit Real numbers for all real-world problems. Direct control over ALL System resources, including interrupts. Comprehensive library support; simple Machine Code interface; step-by-step tracer for instant debugging. 5000 page Manual with tutorial guide. F, CCF - $198.00

WHIMSICAL from **Whimsical Developments** -- Now supports **Real Numbers.** "Structured Programming" WITHOUT losing the Speed and Control of Assembly Language! Single-pass Compiler features unified, user-defined I/O; produces ROMable Code; Procedures and Modules (including pre-compiled Modules); many "Types" up to 32 bit Integers, **6-digit Real Numbers**, unlimited sized Arrays (vectors only); Interrupt handling; long Variable Names; Variable Initialization; Include directive; Conditional compiling; direct Code insertion; control of the Stack Pointer; etc. Run-Time subroutines inserted as called during compilation. Normally produces 10% less code than PL/9. F and CCF - $195.00

C Compiler from **Windrush Micro Systems** by James McCosh. Full C for FLEX except bit-fields, including an Assembler. Requires the TSC Relocating Assembler if user desires to implement his own Libraries. F and CCF - $295.00

C Compiler from **Introl** -- Full C except Doubles and Bit Fields, streamlined for the 6809. Reliable Compiler; FAST, efficient Code. More UNIX Compatible than most.
FLEX, CCF, OS-9 (Level II ONLY), U - $575.00

PASCAL Compiler from **Lucidata** -- ISO Based P-Code Compiler. Designed especially for Microcomputer Systems. Allows linkage to Assembler Code for maximum flexibility.
F and CCF 5" - **$99.95** F 8" **$99.95**

PASCAL Compiler from **OmegaSoft** (now **Certified Software**) -- For the **PROFESSIONAL**; ISO Based, Native Code Compiler. Primarily for Real-Time and Process Control applications. Powerful; Flexible. Requires a "Motorola Compatible" Relo. Assb. and Linking Loader.
F and CCF - $425.00 One Year Maint. - $100.00

K-BASIC from **LLOYD I/O** -- A "Native Code" **BASIC Compiler** which is now **Fully TSC XBASIC compatible.** The compiler compiles to Assembly Language Source Code. A NEW, **streamlined**, Assembler is now included allowing the assembly of LARGE Compiled K-BASIC Programs. Conditional assembly reduces Run-time package.
FLEX, CCF, OS-9 Compiler with Assembler - $199.00

CRUNCH COBOL from **Compusense Ltd.** -- Supports large subset of ANSII Level I COBOL with many of the useful Level 2 features. Full FLEX File Structures, including Random Files and the ability to process Keyed Files. Segment and link large programs at runtime, or implemented as a set of overlays. The System requires 56K and CAN be run with a single Disk System.
FLEX, CCF; Normally $199.00
Special Introductory Price (while in effect) -- **$99.95**

FORTH from **Stearns Electronics** -- A CoCo FORTH Programming Language. **Tailored to the CoCo!** Supplied on Tape, transferable to disk. Written in FAST ML. Many CoCo functions (Graphics, Sound, etc.). Includes an Editor, Trace, etc. Provides CPU Carry Flag accessibility, Fast Task Multiplexing, Clean Interrupt Handling, etc. for the "Pro". Excellent "Learning" tool!
Color Computer ONLY - $50.95

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

!!! Please Specify Your Operating System & Disk Size !!!

GENERAL

SOFTWARE DEVELOPMENT

BasicO9 IRef from Southeast Media -- This **BasicO9 Cross Reference** Utility is a BasicO9 Program which will produce a "pretty printed" listing with each line numbered, followed by a complete cross referenced listing of all variables, external procedures, and line numbers called. Also includes a **Program List Utility** which outputs a fast "pretty printed" listing with line numbers. Requires BasicO9 or RunB.
O & CCO obj. only -- $39.95; w/ Source - $79.95

Lucidata PASCAL UTILITIES (Requires LUCIDATA Pascal ver 3)

XREF -- produce a Cross Reference Listing of any text; oriented to Pascal Source.

INCLUDE -- Include other Files in a Source Text, including Binary; unlimited nesting capabilities.

PROFILER -- provides an Indented, Numbered, "Structogram" of a Pascal Source Text File; view the overall structure of large programs, program integrity, etc. Supplied in Pascal Source Code; requires compilation.

F, CCF --- **XREF Utility** 5" - $40.00, 8" - $50.00

DUB from Southeast Media -- A **UniFLEX** "basic" De-Compiler. Re-Create a Source Listing from UniFLEX **Compiled** basic Programs. Works w/ ALL Versions of 6809 UniFLEX basic. U - $219.95

FULL SCREEN FORMS DISPLAY from Computer Systems Consultants -- TSC Extended BASIC program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.
F and CCF, U - $25.00, w/ Source - $50.00

DISK UTILITIES

OS-9 YDist from Southeast Media -- For **Level I** only. Use the **Extended Memory** capability of your SWTPC or Gimix CPU card for similar format DAT) for FAST Program Compiles, CMD execution, high speed inter-Process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increments up to 960K. Some Assembly Required.
-- **Level I ONLY** -- OS-9 obj. only - $79.95; w/ Source - $149.95

O-F from Southeast Media -- Written in BASICO9 (with Source), includes: **REFORMAT**, a BASICO9 Program that reformats a chosen amount of an **OS-9** disk to **FLEX** Format so it can be used normally by FLEX; and FLEX, a BASICO9 Program that does the actual read or write function to the special **O-F Transfer Disk**; user-friendly menu driven. Read the FLEX Directory, Delete FLEX Files, Copy both directions, etc. FLEX users use the special disk just like any other FLEX disk. **SPECIAL 60 DAY OFFER** O-$39.95

COPYMULT from Southeast Media -- Copy **LARGE** Disks to several **smaller disks.** FLEX utilities allow the backup of **ANY size disk** to any SMALLER size diskettes (Hard Disk to floppies, 8" to 5", etc.) by simply inserting diskettes as requested by **COPYMULT.** No fooling with directory deletions, etc. **COPYMULT.CMD** understands normal "copy" syntax and keeps up with files copied by maintaining directories for both host and receiving disk system. Also includes **BACKUP.CMD** to download any size "random" type file; **RESTORE.CMD** to restructure copied "random" files for copying, or recopying back to the host system; and **FREELINK.CMD** as a "bonus" utility that "relinks" the free chain of floppy or hard disk, eliminating fragmentation.
Completely documented Assembly Language Source files included.
ALL 4 Programs (FLEX, 8" or 5") $99.50

COPYCAT from Lucidata -- Pascal **NOT** required. Allows reading TSC Mini-FLEX, SSB DOS68, and Digital Research CP/M Disks while operating under FLEX 1.0, FLEX 2.0, or FLEX 9.0 with 6800 or 6809 Systems. COPYCAT will not perform miracles, but, between the program and the manual, you stand a good chance of accomplishing a transfer. Also includes some Utilities to help out. Programs supplied in **Modular** Source Code (Assembly Language) to help solve unusual problems.
F and CCF 5" - $50.00 F 8" - $65.00

FLEX DISK UTILITIES from Computer Systems Consultants -- Eight (8) different Assembly Language (w/ Source Code) FLEX Utilities for every FLEX Users Toolbox: **Copy** a File with **CRC Errors**; **Test** Disk for errors; **Compare** two **Disks**; a fast **Disk Backup** Program; **Edit** Disk Sectors; **Linearize** Free-Chain on the Disk; print **Disk Identification**; and **Sort** and **Replace** the Disk Directory (in sorted order). -- **PLUS** -- Ten XBASIC Programs including: A **BASIC Renumberer** with **EXTRAs** over "RENUM" like check for missing label definitions, processes Disk to Disk instead of in Memory, etc. Other programs **Compare, Merge,** or **Generate Updates** between two BASIC Programs, check BASIC **Sequence Numbers**, compare two **unsequenced** files, and 5 Programs for establishing a **Master Directory** of several Disks, and sorting, selecting, updating, and printing paginated listings of these files. A BASIC Cross-Reference Program, written in Assembly Language, which provides an X-Ref Listing of the Variables and Reserved **Words in TSC BASIC, XBASIC,** and **PERCOMPILER BASIC Programs.**
ALL Utilities **Include Source** (either BASIC or A-L- Source Code).
F and CCF - $50.00
BASIC Utilities ONLY for **UniFLEX** — $30.00

COMMUNICATIONS

CMODEM Telecommunications Program from Computer Systems Consultants, Inc. -- Menu-Driven; supports Dumb-Terminal Mode, Upload and Download in non-protocol mode, and the CP/M "Modem7" Christensen protocol mode to enable communication capabilities for almost any requirement. Written in "C".
FLEX, CCF, OS-9, UniFLEX; with complete Source - $100.00
without Source - $50.00

XDATA from Southeast Media -- A **COMMUNICATION** Package for the UniFLEX Operating System. Use with CP/M, Main Frames, other UniFLEX Systems, etc. Verifies Transmission using checksum or CRC; Re-Transmits bad blocks, etc. U - $299.99

GAME

RAPIER - 6809 **Chess** Program from **Southeast Media** -- Requires FLEX and Displays on **Any** Type Terminal. Features: Four levels of play. Swap side. Point scoring system. Two display boards. Change skill level. Solve Checkmate problems in 1-2-3-4 moves. Make move and swap sides. Play white or black. This is one of the strongest CHESS programs running on any microcomputer, estimated USCF Rating 1600+ (better than most 'club' players at higher levels). F and CCF - $79.95

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

!!! Please Specify Your Operating System & Disk Size !!!

BUSINESS

WORD PROCESSING

SCREDITOR III from Windrush Micro Systems -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!

6800 or 6809 FLEX or SSB DOS, OS-9 - $175.00

STYLO-GRAPH from Great Plains Computer Co. -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/STAR-DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.
NEW PRICES —> CCF and CCO - $99.95, F or O - $179.95, U - $299.95

STYLO-SPELL from Great Plains Computer Co. -- Fast Computer Dictionary. Complements Stylograph.
NEW PRICES —> CCF and CCO - $69.95, F or O - $99.95, U - $149.95

STYLO-MERGE from Great Plains Computer Co. -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.
NEW PRICES —> CCF and CCO - $59.95, F or O - $79.95, U - $129.95

JUST from Southeast Media -- Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the Display. Use the FPRINT.CMD supplied for producing multiple copies of the "Formatted" Text on the Printer INCLUDING IMBEDDED PRINTER COMMANDS (very useful at other times also, and worth the price of the program by itself). "User Configurable" for adapting to other Printers (comes set up for Epson MX-80 with Graftrax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line, includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with PAT or any other editor.

* Now supplied as a two disk set:
Disk #1: JUST2.CMD object file, JUST2.TXT PL9 source: FLEX - CC
Disk #2: JUSTSC object and source in C: FLEX - OS9 - CC

The JTSC and regular JUST C source are two separate programs. JTSC compiles to a version that expects TSC Word Processor type commands, (.pp .sp .ce etc.) Great for your older text files.

The C source compiles to a standard syntax JUST.CMD object file. Using JUST syntax (.p .u .y etc.) With all JUST functions plus several additional printer formatting functions. Reference the JUSTSC C source. For those wanting an excellent BUDGET PRICED word processor, with features some of the others have. This is it!

Disk (1) - PL9 FLEX Version only - F & CCF - $49.95
Disk Set (2) - F & CCF & OS9 (C version) - $69.95

SPELL8 "Computer Dictionary" from Southeast Media -- OVER 120,000 words! Look up a word from within your Editor or Word Processor (with the SPH.CMD Utility which operates in the FLEX UCS). Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELL8 first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELL8 also allows the use of Small Disk Storage systems.

!! SPECIAL LIMITED TIME OFFER !! F and CCF - $99.95

DATA BASE - ACCOUNTING

XDMS from Westchester Applied Business Systems -- Powerful DBMS; M.L. program will work on a single sided 5" disk, yet is F-A-S-T. Supports Relational, Sequential, Hierarchical, and Random Access File Structures; has Virtual Memory capabilities for Giant Data Bases. XDMS Level I provides an "entry level" System for defining a Data Base, entering and changing the Data, and producing Reports. XDMS Level II adds the POWERFUL "GENERATE" facility with an English Language Command Structure for manipulating the Data to create new File Structures, Sort, Select, Calculate, etc. XDMS Level III adds special "Utilities" which provide additional ease in setting up a Data Base, such as copying old data into new Data Structures, changing System Parameters, etc.

XDMS System Manual - $24.95 XDMS Lvl I - F & CCF - $129.95
XDMS Lvl II - F & CCF - $199.95
XDMS Lvl III - F & CCF - $269.95

ACCOUNTING PACKAGES -- Great Plains Computer Co. and Universal Data Research, Inc. both have Data Base and Business Packages written in TSC XBASIC for FLEX, CoCo FLEX, and UniFLEX.

MISCELLANEOUS

TABULA RASA SPREADSHEET from Computer Systems Consultants -- TABULA RASA is similar to DESKTOP/PLAN; provides use of tabular computation schemes used for analysis of business, sales, and economic conditions. Menu-driven; extensive report-generation capabilities. Requires TSC's Extended BASIC.

F and CCF, U - $50.00, w/ Source - $100.00

DYRACALC from Computer Systems Center -- Electronic Spreed Sheet for the 6809.

F, SPECIAL CCF and OS9 - $200.00, U - $395.00

FULL SCREEN INVENTORY/MRP from Computer Systems Consultants -- Use the Full Screen Inventory System/Materials Requirement Planning for maintaining inventories. Keeps item field file in alphabetical order for easier inquiry. Locate and/or print records matching partial or complete item, description, vendor, or attributes; find backorder or below stock levels. Print-outs in item or vendor order. MRP capability for the maintenance and analysis of Hierarchical assemblies of items in the inventory file. Requires TSC's Extended BASIC.

F and CCF, U - $50.00, w/ Source - $100.00

FULL SCREEN MAILING LIST from Computer Systems Consultants -- The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Locate all records matching on partial or complete name, city, state, zip, or attributes for Listings or Labels, etc. Requires TSC's Extended BASIC.

F and CCF, U - $50.00, w/ Source - $100.00

DIET-TRAC Forecaster from Southeast Media -- An XBASIC program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P G%) or grams of Carbohydrate, Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual. Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individual are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. Provides number of days and daily calendar after weight goal and calorie plan is determined.

F - $59.95, U - $89.95

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

SOUTH EAST MEDIA
5900 Cassandra Smith Rd. Hixson, TN 37343 into (615) 842-4601 CoCo OS-9 FLEX SOFTWARE

Availability Legend —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

*FLEX is a trademark of Technical Systems Consultants
*OS9 is a trademark of MicroWare

!!! Please Specify Your Operating System & Disk Size !!!

```
                0C1C  tbsz    equ    3100
                0010  lsl     equ    16
                000E  dep     equ    14
                001E  fcs     equ    30
                0040  fsb     equ    64
                0010  ips     equ    16
                0050  mslw    equ    fsb+ips+13
                0005  dirsob  equ    5
                0009  rdst    equ    9
                000A  stss    equ    10

0026                          org    $0026

0026                  track   fcb    1
0021                  sector  rmb    1
0022                  badcnt  rmb    1
0023                  drn     rmb    1
0024                  side    rmb    1
0025                  dbsdf   rmb    1
0026                  dunso   rmb    1
0027                  dnsitr  rmb    1
0028                  stemp2  rmb    2
002A                  secent  rmb    2
002C                  fslav1  rmb    2
002E                  fslavf  rmb    2
0030                  drn2    rmb    1          used for 1 of 4 possible
0031                  dens0   rmb    1
0032                  bnest   rmb    1
0033                  pad     rmb    1
0034                  fkfcb   rmb    4
0038                  volmax  rmb    11
0043                  volavl  rmb    2

3000                          org    $3000

3000 20   18          start   bra    format
3002 03 2C 01 3A              fcc    $03,',',01,':',$03
3006 03
3007 20 35 20 69              fcc    ' 5 inch dsd'
3000 6E 63 68 20
300F 64 00 01
3012 20   00          format  bra    formto
3014 BD   CD1E        outln   jsr    pstrng
3017 BD   CD15        outln2  jsr    getchr
301A 84   5F                  anda   #$5f
301C 81   59                  cmpa   #$59
301E 39                       rts
301F 10CE C07F        formto  lds    #stack
3023 8E   3028                ldx    #track
3026 6F   80          clear   clr    0,x+
3028 8C   3034                cmpx   spare1
302B 26   F9                  bne    clear
302D 86   0A                  lda    #seens0

302F 97   31                  sta    dens0
3031 97   33                  sta    sec
3032 86   14                  lda    #secnsl
3035 97   32                  sta    secsl
3037 BD   CD42        12045   jsr    rsthex
303A 1025 000E                lbcs   exit
303E 1F   10                  tfr    x,d
3040 1083 0003                cmpd   #3
3044 1022 0004                lbhi   exit
3048 8E   0100                ldx    #secrk
304B C7   03                  stb    3,x
304D 07   23                  stb    <drn
304F 86   01                  lda    #1
3051 59                       rolb
3052 27   04                  beq    L3057
3054 48             L3053     asla
3055 5A                       decb
3056 26   FC                  bne    L3053
3058 97   30          L3057   sta    <drn2
305A 0E   3443                ldx    #sures
305D BD   B5                  jsr    outln
305F 26   6D                  bne    exit
3061 0E   347E                ldx    #scrds
3064 BD   CD1E                jsr    pstrng
3067 8E   0102                ldx    #secrk+2
306A 6F   84                  clr    0,x
306C 5F                       clrb
306D BD   CD39                jsr    buldsc
3070 86   3F                  lda    #'?
3072 BD   CD18                jsr    outchr
3075 86   20                  lda    #$20
3077 BD   CD18                jsr    outchr
307A BD   98                  bsr    getin2
307C 26   4E                  bne    exit
307E 0F   25                  clr    <dbsdf
3080 8E   342F                ldx    #dbst
3083 BD   0F                  bsr    pulsm
3085 26   06                  bne    form25
3087 0C   25                  inc    <dbsdf
3089 86   14                  lda    #sbsecs
308B 97   33                  sta    sec
308D 0F   26          form25  clr    <dsmsb
308F 0F   27                  clr    <dnsitr
3091 0E   3452                ldx    #dstr
3094 17   FF70                lbsr   secsz
3097 26   02                  bne    form26
3099 0C   26                  inc    <dsmsb
309B 0E   34F3        form26  ldx    #nstr
309E BD   CD1E                jsr    pstrng
30A1 BD   CD18                jsr    inbuff
30A4 8E   0034                ldx    #fkfcb
30A7 BD   CD20                jsr    getfil
30A9 8E   3501        form27  ldx    #hmstr

3000 BD   CD1E                jsr    pstrng
3000 BD   CD18                jsr    inbuff
3003 BD   CD48                jsr    isdec
3006 25   F2                  bcs    form27
3008 9F   4D                  stx    xblnvo
300A BD   CD24                jsr    pcrlf
300D 0E   0100                ldx    #secrk
30C0 BD   BL89                jsr    drpst
30C3 27   15                  beq    forOut
30C5 8E   3468                ldx    #wmsg
30C8 C5   40                  bitb   #$40
30CA 26   03                  bne    ezsl2

30CC 8E   34A5        exsl    ldx    #oberss
30CF BD   CD1E        exsl2   jsr    pstrng
30D2 BD   0403        exsl3   jsr    fmscls
30D5 1C   CF                  cli
30D7 7E   C003                jmp    warms

30DA 18   10          format  orcc   #$10
30DC 8F   28                  clr    <track
30DE 8F   24          form3   clr    <side
30E0 8F   21                  clr    <sector
30E2 BD   40                  bsr    lrhhd
30E4 8E   0124        form32  ldx    #work+33  total trackheader + 33
30E7 BD   27                  tst    <dnsitr
30E9 27   03                  beq    formd
30EB 8E   033A                ldx    #work+58  total track header + 30
30EE BD   6E          formd   bsr    dosec
30F0 8C   21                  inc    <sector
30F2 96   21                  lda    <sector
30F4 BB   24                  tst    <side
30F6 26   04                  bne    form45
30F8 91   31                  cmpa   secs0
30FA 20   02                  bra    form46
30FC 91   32          form45  cmpa   secs1
30FE 26   EE          form46  bne    formd
3100 96   28                lda    <track
3102 D6   24                  ldb    <side
3104 BD   DE18                jsr    dsprt
3107 BD   3388                jsr    wrtirk
310A BD   25          form3   lst    <dbsdf
310C 27   08                  beq    form6
310E BD   24                  tst    <side
3110 26   04                  bne    form6
3112 03   24                  com    <side
3114 20   CE                  bra    form32
3116 0C   28          form6   inc    <track
3118 BD   3762                jsr    spotch
311B 96   28                lda    <track
311D 81   28                  cmpa   #sectrk
311F 26   08                  bne    form7
3121 16   00F4                lbra   setup

3124 0E   0100        lrhhd   ldx    #secrk
3127 BD   27                  tst    <dnsitr
3129 26   11                  bne    srhdd
312B C6   FF                  ldb    #$ff
312D E7   88          lrhds1  stb    0,x+
312F 8C   0D1C                cmpx   #sabend
3132 26   F9                  bne    lrhds1
3134 0C   0110                ldx    #secrb+$$t
3137 4F                       clra
3138 C6   06                  ldb    #6
313A 20   15                  bra    lrhdd2
313C C6   4E          srhdd   ldb    #$4e
313E E7   80          lrhdd1  stb    0,x+
3140 8C   1938                cmpx   #dskend
3143 26   F9                  bne    lrhdd1
3145 0E   0120                ldx    #$$ts2+secrb
3148 4F                       clra
3149 C6   0C                  ldb    #12
314B BD   00                  bsr    sst
314D 86   F6                  lda    #$f6
314F C6   03                  ldb    #3
3151 BD   05          lrhdd2  bsr    sst
3153 86   FC                  lda    #$fc
3155 A7   04                  sta    4,x
3157 39                       rts
3158 A7   00          sst     sta    0,x+
315A 5A                       decb
315B 26   F8                  bne    sst
315D 39                       rts
315E 4F               dosec   clra
315F BD   27                  tst    <dnsitr
3161 26   04                  bne    dosec1
3163 C6   06                  ldb    #6
3165 20   00                  bra    dosec2
3167 C6   0C          dosec1  ldb    #12       clear 12 bytes
3169 BD   CD                  bsr    sst
316B 86   F5                  lda    #$f5
316D C6   03                  ldb    #3
316F BD   E7          dosec2  bsr    sst
3171 86   FE                  lda    #$fe
3173 A7   00                  sta    0,x
3175 96   20                  lda    <track
3177 A7   00                  sta    0,x
3179 06   27                  ldb    <dnsitr
317B 27   04                  beq    dosec3
317D 06   24                  ldb    <side
317F C4   01                  andb   #$01
3181 E7   00          dosec3  stb    0,x+
3183 06   21                  ldb    <sector
3185 100E 31C0                ldy    #$$cnoo
3189 BD   27                  tst    <dnsitr
318B 27   04                  beq    dosedd
318D 100E 31F4                ldy    #dscnoo

3191 E6   05          dosec4  ldb    5,x
3193 E7   00                  stb    0,x+
3195 01   33                  com    sec
3197 26   09                  bne    dosec7
3199 4C                       inca
319A 5F                       clrb
319B 01   28                  cmpa   #sectrk
319D 26   03                  bne    dosec7
319F 4F                       clra
31A0 C6   FF                  ldb    #$ff
31A2 5C               dosec7  incb
31A3 34   06                  pshs   a,b
31A5 86   01                  lda    #1
31A7 A7   00                  sta    0,x+
31A9 86   F7                  lda    #$f7
31AB A7   00                  sta    0,x+
31AD BD   27                  tst    <dnsitr
31AF 26   07                  bne    dosec8
31B1 30   88                  leax   -$$,x
31B3 4F                       clra
31B4 C6   06                  ldb    #6
31B6 20   BC                  bra    dosec9
31B8 30   00 16       dosec8  leax   22,x
31BB 4F                       clra
31BC C6   0C                  ldb    #12
```

```
310E 4D    98            tsx     sat
31C0 86    15            lda     #$f5
31C2 C6    03            ldb     #3
31C4 BD    92      dosec9  jsr     sat
31C6 86    FB            lda     #$fb
31C8 A7    88            sta     8,x+
31CA 35    04            puls    a,b
31CC C0    01            subd    #1,++
31CE 4F                  clra
31CF C6    FE            ldb     #$fd
31D1 BD    95            jsr     sat
31D3 86    F7            lda     #$f7
31D5 A7    88            sta     8,x+
31D7 3B    0E            leax    fee.x      eee is 14
31D9 8D    27            bsr     dosecd
31DB 2F    07            ble     dosec4
31DD 3C    0B            leax    eee-3,x
31DF 39          dosec6  rts
31E0 01 06 0a 09 sssss   fcb     $01,$06,$04,$09,$02,$07,$05,$0a

31E4 02 07 05 0B
31E8 03 0B 12 0B         fcb     $03,$0b,$12,$0b,$18,$0a,$13,$0c

31EC 10 0C 13 0C
31F0 11 0F 14 0D         fcb     $11,$0f,$14,$0d

31F4 09 04 07 0B dscaaa  fcb     $01,$04,$07,$0b,$06,$10,$02,$05

31F8 0D 10 02 05
31FC 08 0B 0E 11         fcb     $08,$0b,$0e,$11,$0d,$06,$09,$0c

3200 03 06 09 0C
3204 0F 12 16 19         fcb     $0f,$12,$16,$19,$1c,$1f,$22,$13

3208 1C 1F 22 13
320C 17 1A 1D 20         fcb     $17,$1a,$1d,$20,$23,$14,$18,$1b

3210 23 14 18 1B
3214 1E 21 24 15         fcb     $1e,$21,$24,$15

3218 96    33      solve   ldb     #$33       read all sectors for errors
321A 86    27              lda     #seclrk-1
321C DD    2E              std     clistavl
321E 3D                    mul
321F DD    28              std     cseccnt
3221 8E    0101            ldx     #first
3224 9F    2D              stx     cfstavl
3226 96    33              lda     cdrn
3228 B7    0107            sta     work+3
322B 4F                    clra
322C 97    32              sta     cbadcnt
322E 97    20              sta     ctrack
3230 97    27              sta     cdensity
3232 4C                    inca
3233 97    21              sta     csector
3235 86    0B              lda     #seccs0
3237 97    31              sta     secs0
3239 86    14              lda     #seccs1
323B 97    32              sta     secs1
323D DB    25              tst     cdbsdf
323F 26    02              bne     setse1
3241 86    0A              lda     #seccs2
3243 97    33      setse1  sta     secs
3245 BD    10      setse2  jsr     cshiow
3247 26    3C              bne     rshsse
3249 8F    22              clr     cbadcnt
324B DC    28      setse4  ldd     ctrack
324D BD    2a              jsr     firsec
324F 1027  0003            lbeq    dolrh
3253 DD    28              std     ctrack
3255 20    EC              bra     setse2
3257 8E    0100    chksse  ldx     #work
325A DC    28              ldd     ctrack
325C ED    88 1E           std     fcs,x
325F 7E    32EF            jmp     readss
3262 D6    26      switch  ldb     cdensa
3264 27    12              beq     switch2
3266 D7    27              stb     cdensity
3268 C6    12              ldb     #dbacs0
326A D7    31              stb     secs0
326C C6    24              ldb     #dbacs1
326E D7    32              stb     secs1
3270 DB    25              tst     cdbsdf
3272 26    02              bne     switch1
3274 C6    12              ldb     #dacs0
3276 D7    33      switch1 stb     secs
3278 39         switch2 rts
3279 81    33      firsec  cmpb    secs

327B 26    04              bne     firsec4
327D 4C                    inca
327E BD    E2              bsr     switch
3280 5F                    clrb
3281 5C         firsec4 incb
3282 C1    28              cmpb    #seclrk
3284 39                    rts
3285 0C    22      reesec  inc     cbadcnt
3287 27    0d              beq     reesc1
3289 96    28              lda     ctrack
328B 26    0C              bne     reesc2
328D D6    21              ldb     csector
328F C1    05              cmpb    #dircyc    read directory
3291 22    06              bhi     reesc7
3293 8E    3e39    reesc1  ldx     #fstore
3296 7E    30CF            jmp     exit2
3299 8E    0100    reesc2  ldx     #work
329C DC    2C              ldd     cfstavl
329E 1093  28              cmpd    ctrack
32A1 26    06              bne     reesc3
32A3 BD    04              bsr     firsec
32A5 DD    2C              std     cfstavl
32A7 20    27              bra     reesc8
32A9 DC    28      reesc3  ldd     ctrack
32AB B0    32              suba    cbadcnt
32AE 27    02              beq     reesc4
32B0 20    03              bra     reesc5
32B2 4A         reesc35 deca
32B3 D6    33      reesc4  ldb     sec
32B5 C0    88 1E           subd    fcs,x
32B8 BD    36              bsr     readss
32BA 26    B0              bne     reesc1
32BC EC    88 4B           ldd     fsb,x
32BF BD    89              bsr     flesse
32C1 26    B7              bne     reesc4
32C3 EC    88 1E           ldd     fcs,x
32C6 DD    2C              std     clistavl
32C8 4F                    clra
32C9 C0    88 4B   reesc4  subd    fsb,x
32CC BD    28              bsr     writss
32CE 26    C3              bne     reesc1
32D0 9E    2A      reesc6  ldx     cseccnt
32D2 3B    1F              leax    -1,x
32D4 9F    2A              stx     cseccnt
32D6 8C    3400            cmpx    #bad$0
32D9 8B    CD1E            jsr     outres
32DC 8E    0020            ldx     #lrock
32DF BD    CD3C            jsr     outhex
32E2 86    2B              lda     #$2B
32E4 BD    CD1A            jsr     outchr
32E7 3B    81              leax    1,x
32E9 BD    CD3C            jsr     outhex

32EC 7E    3248            jmp     setend
32EF 8E    0100    readss  ldx     #work
32F2 D6    89              ldb     #rdss
32F4 A7    84              sta     0,x
32F6 7E    0406            jmp     fas
32F9 8E    0100    writss  ldx     #work
32FC D6    8B              ldb     #wtss
32FE A7    84              sta     0,x
3300 BD    0406            jsr     fas
3303 27    EA              beq     readss
3305 39                    rts
3306 0F    27      dolrk   clr     cdensity
3308 8E    0100            ldx     #work
330B 6F    88 1C           clr     fcs,x
330E 86    03              lda     #3
3310 A7    88 1F           sta     fcs+1,x
3313 BD    DA              bsr     readss
3315 26    48              bne     dolrk4
3317 8E    0100            ldx     #work
331A 6F    88 48           clr     fsb,x
331D 6F    88 41           clr     fsb+1,x
3320 DC    2C              ldd     cfstavl
3322 C0    88 50           subd    #$50
3325 DC    2C              ldd     cfstavl
3327 C0    88 5F           subd    #$5f
332A DD    28              std     cseccnt
332C CB    88 61           addb    #$61
332F 86    27              lda     #maxtrk-1
3331 B6    31              ldb     secs0
3333 BD    25              tst     cdbsdf
3335 26    97              bne     dblrk2
3337 B6    33              ldb     secs1
3339 CB    88 66   dolrk2  addb    #$16+9,x
333C FC    CC0E            ldd     sruded
333F CB    88 63           addb    #$16+6,x
3342 B6    CC10            lda     sruded+2
3345 A7    88 65           sta     $16+8,x
3348 C6    09              ldb     #13
334A 10BE  8030            ldy     $rolaaa
334E 8E    0100            ldx     #work
3351 A6    A0      dolr33  lda     0,y+
3353 A7    88 50           sta     fsb+$19,x
3356 3B    81              leax    1,x
3358 5A                    decb
3359 26    F6              bne     dolr32
335B BD    9C              bsr     writss
335D 27    03              beq     dirlnl
335F 7E    3293    dolrk4  jmp     reesel
3362 8E    0100    dirlnl  ldx     #work
3365 B6    88              lda     #$aaaa8
3367 BB    35              tst     cdbsdf
3369 27    02              beq     dirlnl
336B B6    14              lda     #$aaa5l

336D A7    88 1F   dirlnl  sta     fcs+1,x
3370 BD    32EF            jsr     readss
3373 26    EA              bne     dolrk4
3375 8E    0100            ldx     #work
3378 6F    88 48           clr     fsb,x
337B 6F    88 41           clr     fsb+1,x
337E BD    32F9            jsr     writss
3381 26    DC              bne     dolrk4
3383 8E    C100    dobool  ldx     #$100
3386 96    26              lda     cdensa
3388 B7    C107            sta     dns
338B 4F                    clra
338C C6    01              ldb     #1
338E BD    0E03            jsr     dwrite     do write single sector
3391 26    CC              bne     dolrk4
3393 8E    0100            ldx     #work
3396 86    10              lda     #10
3398 A7    84              sta     0,x
339A BD    0406            jsr     fas
339D 26    C8              bne     dolrk4
339F B6    B7              lda     #$7
33A1 A7    84              sta     0,x
33A3 BD    0406            jsr     fas
33A6 26    B7              bne     dolrk4
33A8 8E    34CA            ldx     #csolls
33AB BD    CD1E            jsr     outrnd
33AE 8E    0115            ldx     #work+21
33B1 5F                    clrb
33B2 BD    C039            jsr     outdec
33B5 7E    3302            jmp     exit3

33B8 B8    58      writck  ebr     $34f4
33BA 8E    F3E3            ldx     #$f3e3     reverse of 3100 single dens.
33BD BD    27              tst     cdensity   is ddens?
33BF 27    03              beq     $34ca
33C1 8E    E7C7            ldx     #$e7e7     reverse of 6200
33C4 9F    2B      $34ca   stx     cstepo2
33C6 8E    0100            ldx     #work
33C9 BD    9F F012         jsr     [$ro7      set root address
33CC BB    1B              org     $01b
33CE B7    F04B            sta     extadr
33D1 BF    F04A            sta     ch8adr
33D4 F3    F04A            add     ch8adr
33D7 F3    F051            add     ch8adrof
33DA 9E    28              ldx     cxtemp2
33DC BF    F053            stx     ch8brt
33DF C6    FE              ldb     #$fe
33E1 F7    F01B            stb     ch8cko
33E4 C6    FE              ldb     #$fe
33E6 B6    FA              lda     $1f4
33E9 F7    F01A            stb     orlaan
```

```
33EC B7   F020          sta    fdstat
33EF 7B   F816  l3ed7   tst    ch@cbe

33F2 2B   FB            bmi    l3ed7
33F4 8E   FFFF          ldx    #1ffff
33F7 BF   F814          stx    xricon
33FA 20   0A            bra    l34ee
33FC B7   F023  l34e4   sta    fdcntl
33FF 86   18            lda    #$18
3401 B7   F020          sta    fdstat
3404 8D   08            bsr    l34f6
3406 B6   F020  l34ee   lda    fdstat
3409 85   01            bita   #1
340B 26   F9            bne    l34ee
340D 39                 rts
340E 8D   00    l34f6   bsr    l34f8
3410 8D   06    l34r8   bsr    l34fe
3412 8D   00    l34fa   bsr    l34fc
3414 39         l34fc   rts
3415 8D   24    l34fd   bsr    fside
3417 26   06            bne    l3507
3419 D6   38            ldb    fdrn2
341B CA   08            orb    #$08
341D 20   04            bra    l350b
341F D6   38    l3507   ldb    fdrn2
3421 CA   90            orb    #$9 8
3423 8D   27    l350b   bsr    fdensity
3425 26   02            bne    l3511
3427 CA   20            orb    #$20
3429 53         l3511   comb
342A F7   F024          stb    drvsel
342D 20   BF            bra    l34f6

342F 64 6F 75 62  d0st    fcc    "double sided disk? "
3433 6C 65 20 73
3437 69 64 65 64
343B 20 64 69 73
343F 6B 3F 20 04
3443 61 72 65 20  bprs    fcc    "are you sure? "
3447 79 6F 75 20
344B 73 75 72 65
344F 3F 20 04
3452 64 6F 75 62  ddstr   fcc    "double density disk? "
3456 6C 65 20 64
345A 65 6E 73 69
345E 74 79 20 64
3462 69 73 6B 3F
3466 20 04
3468 20 20 20 20  wpst    fcc    "-- disk is protected! "
346C 69 73 20 70
3470 72 6F 74 65
3474 63 74 65 64
3478 21 04
347C 73 63 72 61  scrds   fcc    "scratch disk in drive "
3480 74 63 68 20
3484 64 69 73 6B
3488 20 69 6E 20
348C 64 72 69 76
3490 65 20 04
3495 66 61 74 61  fstrs   fcc    "fatal error --- "
3499 6C 20 65 72
349D 72 6F 72 20
34A1 20 20 20 20
34A5 66 6F 72 6D  abortm  fcc    "formatting aborted"
34A9 61 74 74 69
34AD 6E 67 20 61
34B1 62 6F 72 74
34B5 65 64 04
34B8 20 20 20 62  badsct   fcc    "-- bad sector at "
34BC 61 64 20 73
34C0 65 63 74 6F
34C4 72 20 61 74
34C8 20 04
34CA 66 6F 72 6D  asmtlk   fcc    "formatting assembler"
34CE 61 74 74 69
34D2 6E 67 20 63
34D6 6F 6D 70 6C
34DA 65 74 65 0D
34DE 0A
34DF 00 00 00 74          fcc    nul.nul.nul."total sector"
34E3 6F 74 61 6C
34E7 20 73 65 63
34EB 74 6F 72
34EE 73 20 30 20          fcc    "s = "
34F2 04
34F3 76 6F 6C 75  nmstr   fcc    "volume name? "
34F7 6D 65 20 6E
34FB 61 6D 65 3F
34FF 20 04
3501 76 6F 6C 75  nustr   fcc    "volume number? "
3505 6D 65 20 6E
3509 75 6D 62 65
350D 72 3F 20 04

              C300  sctbuf  equ    $c300     data sector buffer
              FDRE  rdata   equ    $fdee
              FD14  cntrl   equ    $fd14

              FD20  cmdreg  equ    $fd20     command register
              FD21  trkreg  equ    cmdreg+1  track register
              FD22  secreg  equ    cmdreg+2  sector register
              FD23  datreg  equ    cmdreg+3  data register
              FD24  drvreg  equ    cmdreg+4  drive select register

                    * start of utility

C100                        org    $c100

C100 20   00        load    bra    load0

C102 00 00                  fcb    0,0
C104 05             lrz     fcb    5         lrz 4 times
C105 00             trk     fcb    0         file start track
C106 00             sct     fcb    0         file start sector
C107 00             dns     fcb    0         density flag 0=sd 1=dd
C108 C1C0           tsadr   fdb    $c1c0     transfer address
C10A 0000           ladr    fdb    0         load address

C10C 10CE C07F      load0   lds    #stack    setup stack
C110 86   FB                lda    #$fb+cmdreg/256
C112 1F   8B                tfr    a,dp
                      007 B         setdp  cmdreg/256
C114 86   00                lda    #restor   restore to trk0
C116 17   0086              lbsr   xcmd
C119 7A   C104              dec    lrz       lrz again
C11C 26   06                bne    again
C11E 30   8D 00C6           leax   reader,pcr
C122 20   09                bra    error
C124 FC   C105      again   ldd    trk       setup starting trk & sct
C127 26   08                bne    linked    disk is linked
C129 30   8D 0081           leax   nolink,pcr
C12D BD   FDAE      error   jsr    zdata     print message
C130 7E   F814              jmp    cntrl     return to monitor

C133 F8   C300      linked  std    sctbuf
C026 108E C400              ldy    #sctbuf+256

                    * perform actual file load

C138 80   24        load1   bsr    getch     get a character
C13C 81   02                cmpa   #$02      data record header?
C13E 27   18                beq    load2     skip if so
C140 81   16                cmpa   #$16      err address header?
C142 26   F6                bne    load1     loop if neither
C144 8D   28                bsr    getch     get transfer address

C146 B7   C100              sta    tsadr
C149 8D   25                bsr    getch
C14B B7   C109              sta    tsadr+1
C14E 20   EA                bra    load1     continue load
C150 8D   1E        load2   bsr    getch     get load address
C152 B7   C108              sta    ladr
C155 8D   19                bsr    getch
C157 B7   C109              sta    ladr+1
C15A 8D   14                bsr    getch     get byte count
C15C 1F   89                tfr    a,b       put in b
C15E 27   DA                beq    load1     take if count=0
C160 BE   C108              ldx    ladr      get load address
C163 34   14        load3   pshs   b,x
C165 8D   09                bsr    getch     get a data character
C167 35   14                puls   b,x
C169 A7   88                sta    0,x+      put character
C16B 5A                     decb              end of data in record?
C16C 26   F5                bne    load3     take if not
C16E 20   C8                bra    load1     get another record

                    * get character routine - reads a sector if necessary

C170 108C C400      getch   cmpy   #sctbuf+256 out of data?
C174 26   11                bne    getch4    go read character if not
C176 8E   C300      getch2  ldx    #sctbuf   point to buffer
C179 EC   84                ldd    0,x       get forward link
C17B 27   0D                beq    go        if zero, file is loaded
C17D 8D   0F                bsr    read      read next sector
C17F 1026 FF70              lbne   load      start over if error
C183 108E C304              ldy    #sctbuf+4 point past link
C187 A6   80        getch4  lda    0,x+      also, get a character
C189 39                     rts

                    * file is loaded, jump to it

C18A 6E   9F C108   go      jmp    [tsadr]   jump to transfer address

                    * read single sector

              FD24  rdatst  equ    cmdreg+4
              00DF  ddden   equ    %11011111
              007E  drive0  equ    %01111110
              00FF  sel5    equ    %11111111
              00EF  side1   equ    %11101111
              0001  hmsk    equ    1
              0002  drq     equ    2
              00BC  rdcmd   equ    $bc
              0018  restor  equ    $18
              0008  rsetor  equ    $8

C18E 80   1C        read    bsr    xseek     seek track
C190 86   8C                lda    #rdcmd    read sector
C192 97   20                sta    cmdreg    set command

C194 00   42                bsr    dot31     dotor
C196 8E   C300              ldx    #sctbuf   point to sector buffer
C199 20   04                bra    read3

C19B 96   23        read1   lda    datreg    read data
C19D A7   80                sta    ,x+
C19F D6   20        read3   ldb    secreg    set status
C1A1 C5   02                bitb   #drq
C1A3 26   F6                bne    read1     read data
C1A5 C5   01                bitb   #busy     still busy?
C1A7 26   F6                bne    read3
C1A9 C5   1C                bitb   #$1c00111100 errors?
C1AB 39                     rts

C1AC D7   32        xseek   stb    secreg    set sector
C1AE 34   02                pshs   a          save track
C1B0 86   7E                lda    #drive0
C1B2 7D   C107              tst    dns       test density
C1B5 27   04                beq    single
C1B7 C1   12                cmpb   #dbacs10
C1B9 28   04                bra    sideso
C1BB 84   DF        single  anda   #ddden
C1BD C1   08                cmpb   #secs10   see sector side 0
C1BF 23   02        sideso  bls    side0
C1C1 84   EF                anda   #side1    [side1 select side 1
C1C3 97   24        side0   sta    fdcsel    select drive+density
C1C5 35   02                puls   a          get track
C1C7 91   21                cmpa   trkreg    this track?
C1C9 27   0C                beq    dot3h     skip if ok
C1CB 97   23                sta    datreg    set track
C1CD 86   18                lda    #rsetpo   seek
C1CF 97   20        xcmd    sta    cmdreg
C1D1 80   86                bsr    dot31
C1D3 86   20        xwait   ldb    secreg    read status
C1D5 C5   01                bitb   #busy
C1D7 20   FA                bne    xwait
```

```
C1D9 00   00       dot31   bsr    dot12
C1DB 00   00       dot12   bsr    dot5
C1DD 39            dot5    rts

C1DE 6E 6F 20 6C   emlinb  fcc    /no line/
C1E2 69 6E 65
C1E5 00 0A 04              fcb    13,10,4
C1E8 72 65 61 64   reader  fcc    /read error/
C1EC 20 65 72 72
C1F0 6F 72
C1F2 00 0A 04              fcb    13,10,4

                           end    start

0 ERROR(S) DETECTED
```

```
 floppie1.src
 creator for 5" and 8" drives
```

```
*****************************************
*                                       *
*       creator for flex 2.0i3.         *
*                                       *
* correct use, sectors per side when se-*
* lecting drive 2 or 3 which are 5 inch.*
*                                       *
*                                       *
*     [obert Jan van den Bussche        *
*     written: 840809                   *
*                                       *
*****************************************

* include in the startup.txt file:

* set 8:floppie1.bin

DE2I  curdrv  equ    $de2I
DE1E  side    equ    $de1e
DE1F  device  equ    $de1f

DF4C                  org    $df4c
```

```
DF4C 26   06       retrn   bne    stodos    copy code first at $df48
DF4E 60   E4               tst    0,s
DF50 27   02               beq    stodAs
DF52 C6   00               ldb    #0
DF54 F7   DE1E     stodos  stb    >side     store density enable byte
DF57 26   50               bne    stored
DF59 34   02               pshs   a
DF5B B6   DE2I             lda    >curdrv   test for minifloppy
DF5E C6   1E               ldb    #30       8 inch DD
DF60 81   02               cmpa   #$02      if 2 or 3 minifloppy
DF62 25   02               blo    right1    no, skip
DF64 C6   13               ldb    #19       5 inch DD
DF66 35   02       right1  puls   a
DF68 B3   02               subb   $02
DF6A 28   28               bra    verder    overlay full, extend

DF96                       org    $df96

DF96 27   28       verder  beq    stored
DF98 34   02               pshs   a
DF9A B6   DE2I             lda    >curdrv
DF9D C6   1B               ldb    #27       8 inch DD (noraat DD)
DF9F 81   02               cmpa   #$02
DFA1 25   02               blo    right2
DFA3 C6   13               ldb    #19       5 inch DD
DFA5 35   02       right2  puls   a
DFA7 28   0F               bra    stored

DFA9 34   02       stored  pshs   a
DFAB B6   DE2I             lda    >curdrv
DFAE C6   10               ldb    #16       8 inch SD
DFB0 81   02               cmpa   $02
DFB2 25   02               blo    right3
DFB4 C6   08               ldb    #11       5 inch SD
DFB6 35   02       right3  puls   a
DFB8 F7   DE1F     stored  stb    >device   update bez. sectors
DFBB 35   96               puls   a,b,x,pc

                           end

0 ERROR(S) DETECTED
```

Bit Bucket

A 6800 IN THE LABORATORY

Keith Michal
Physiology Department
The Ohio State University
Columbus, Ohio 43210

A computer in a research laboratory is a powerful tool for the collection, analysis and storage of experimental data. Many applications do not need the sophistication and capabilities of laboratory minicomputer and we decided to build an interface for the Southwest Technical Products Corp. 6800 (219 W. Rhapsody, San Antonio, TX) and have used it in our experiments for several years. Biological measurements require a preamplifier to impedance match and amplify the rather small electrical signals generated by living tissues or require transducers to change mechanical, chemical or thermal data into an electrical signal. After the biological data is converted to usable electrical levels, what is required in an interface to make a microprocessor emulate a laboratory computer? Basically, we need the ability to input and output analog voltages, and to input and output binary signals. A list of the interface functions shows these input-output capabilities: 4 channels of analog-to-digital (A/D) conversion, 4 sense line or sense switch inputs, 3 Schmitt trigger inputs, 3 relay or TTL pulse outputs, and 2 digital-to-analog (D/A) channels.

Since most of these functions operate independently, each will be discussed separately as to function, construction and use in the demonstration program that accompanies this article. A summary of the functions is found in Fig. 1. The interface requires the output lines from two parallel interface boards (MP-L) and the interrupt timer board (MP-T) from SWTPc. To save or to read data on tape, a cassette interface unit is used (TC-3, JPC Products, 12021 Paisano Ct. NE, Albuquerque, NM). We mounted the interface inside the computer cabinet because we wanted a portable system with as few interconnected units as possible. A major advantage of mounting inside the cabinet is that the 31 data lines between the computer and the interface are hidden inside the cabinet. All laboratory connections to the interface are made via banana jacks mounted on the rear panel. The voltage outputs from potentiometers used in setting the thresholds for the Schmitt triggers and all input jacks on the back panel are connected to a rear panel mounted rotary switch to facilitate monitoring the different interface voltages on an oscilloscope.

The five integrated circuits and the three relays used in the interface are mounted on a perf board which is attached inside the back panel on 1-1/4 inch stand off mounts directly forward of the banana jacks. The power for these integrated circuits is supplied by the +5 volt supply on one of the parallel interface boards. The relays are powered by the +12 volt computer supply. A ±15 volt modular power supply for the D/A modules was flush mounted inside the front panel just above the power switch. The two D/A modules were mounted on perf boards and these were attached to the front panel above the power supply module. Ribbon cable was used between the computer and the interface.

D/A function (Fig. 1A). From the many available D/A modules, we chose the Datel Systems, Inc. (Model DAE-49) (1020 Turnpike St., Canton, MA). This is a 10 bit converter, however we currently use only 8 bits of this capability to drive a 0 to +5 volt output signal. With these modules we have the capacity to change to a ±5 or ±10 volt output. In use, an 8 bit binary word is stored at the address of the parallel output port connected to the D/A module, which outputs a DC

voltage that is proportional to the magnitude of the binary word. The two D/A modules are used to drive the verticle and the horizontal inputs of an oscilloscope to allow a 256 x 256 point graphics display. In the accompanying program, the labels H and Y indicate the output ports used to drive the D/A modules (see the scope calibration routine at line 1880). The output from one D/A module is used in the A/D conversion discussed below.

The A/D conversion function (Fig. 1B) is basically a voltage comparator circuit. The input voltage is connected to the inverting input of a comparator circuit and the non-inverting input is connected to the output of one of the D/A modules. For analog conversion, the computer program makes successive binary guesses which the D/A module converts to an analog voltage. When this D/A voltage output equals the unknown voltage (as detected by the comparator) the computer stores the current guess as the binary equivalent of the unknown voltage. The analog conversion subroutine is labeled SAM1 (line 2330). The demonstration program has a maximum conversion rate of 3000 analog samples per second. Since an 8 bit binary number contains 2^8 or 256 possible steps, our 5 volt (D/A) output resolves into 256 increments of 19.5 mvolts each (5 volts/256 steps). Conversion is limited to voltages between 0 and 5 volts. Overvoltage protection is provided by a 5 volt zener diode at the input. The interface uses a four comparator integrated chip (LM339) and four input bits of the parallel port at location 8014 to form four A/D converters. The voltage available from a level detector potentiometer can be sampled by one A/D channel and used as a constant in calculations or to provide a variable voltage for display offset, games, etc.

Sense switches or sense lines (Fig. 1C) are a method of inputing a one bit signal to the computer. This bit can be tested and used in a computer program to change program flow by branching to a different part of the program. In our interface circuit, a sense line bit can be activated by a SPDT switch on the front panel of the computer or by grounding a banana jack on the rear panel. This rear panel jack allows remote activation or an interaction between the computer and other laboratory equipment. The interface uses the other half of the PIA port used for A/D conversion at location SNS (8014) as the input port for sense line function. An example of sense switch use is seen in the holding loop (line 1060). If sense switch 1 is activated, program flow is rerouted to the subroutine to clear the display buffer. The effect would be to clear the oscilloscope screen of data and start refilling the buffer with new data. In our application, we have four sense line inputs buffered by the four NAND'S in a 7400 integrated circuit.

The Level Detection Function (Fig. 1D) is used to signal the computer that an analog voltage input has exceeded a preset level. When the input voltage reaches this level, a comparator circuit, functioning as a Schmitt trigger, triggers a monostable pulse which signals the computer. Input voltages are limited to between 0 and 5 volts by a zener diode. The noninverting input of the comparator is connected to a potentiometer mounted on the back panel which sets the trigger voltage level between 0 and 5 volts. The inverting input of the comparator is connected to the signal voltage. When the signal voltage is lower than the voltage set by the potentiometer, the comparator output is low. If the signal voltage exceeds the potentiometer voltage, the comparator output switches high. The transition from low to high is used to trigger an interrupt in one of the parallel ports. This function can be used to count events or to reroute the computer program. The software counting routine, in the interrupt service routine has been used to count sine waves to a frequency of 17,000 Hz. This count (0 to 17000 Hz) was within 1 cycle of the count on a commercial frequency meter. The potentiometer outputs and the signal voltage input are connected to a rotary switch on the back panel whose output is monitored with an oscilloscope in order to adjust the potentiometer level so that the reference level overlaps the input signal and allows the circuit to trigger an interrupt.

Relay or TTL Pulse Function (Fig. 1E) is one bit output capability that can be used to switch on lights or control other types of equipment. The C2 outputs on the SWTPC 6800 parallel interface card are TTL level output bits that can be switched "high" or "low" by the computer program. We have utilized three of these C2 bits to control three reed relays. The 12 volt reed relays are powered by the +12 volt computer power supply. Relays with a coil resistance of 400 ohms or larger must be used or the current rating to the 7407 chip will be exceeded and the 12 volt supply will be overloaded. A switch on the back panel switches the C2 output directly to a banana jack on the rear panel for outputing a TTL level signal (see Fig. 1E).

Scope Intensity Function (Fig. 1F) On our oscilloscope 30 volts applied to the intensity input causes screen blanking or no trace. When a point on the oscilloscope is to be made visible, the 30 volts is removed by grounding a pull up resistor with a pulse from the 7407 chip; remember that the display is a matrix of 256 x 256 points. Blanking improves the clarity of the X-Y display since the intensity is turned off while the beam is moving between display points (see subroutine Bright, line 2150). The thirty volts is input on the rear panel and comes from an external power source. Each oscilloscope requires a different blanking voltage, check your manual to determine the proper level to blank your screen.

Demonstration Program (Listing 1) The accompanying program demonstrates the use of this interface, it asks for directions from the operator, and then enters a waiting loop. While waiting for clock interrupts, the display buffer is displayed. With each clock interrupt, another point is placed in the 256 location data buffer (i.e., at a rate of 1 interrupt per second the buffer would fill in 256 seconds). The buffer wraps around and refills from the left after it is full. Under sense switch control, sampling can be halted when the buffer is full. The display buffer is continuously displayed as a sequential plot on the oscilloscope and can be dumped to or filled from cassette tape. At longer clock intervals (in the counting mode), the latest count can also be displayed numerically on the computer terminal (this display can be enabled or disabled with sense switch 4). An event marker can be inserted into the data string (sense switch 3). As an example, we mark our data to show when a hormone is injected into the brain of an animal. This mark separates the data on preinjection or the control level of neural impulse counts/sec from the response of the nerve cells to the hormone.

Running the program The program first asks for directions by printing on the terminal the following:

"Commands: A E D R M I for info".

Typing "I" causes the following list on commands and functions to be printed:

```
At (:) Enter A, E, D, M or R + #'s
                    A - A/D sample mode
                    E - Event counting mode
                    D - Display storage &
                        calibration
                    M - Go to JPC Monitor
                    R - Set sampling rate
Sense switch function:
                    1 - Clear & restart
                    2 - Hold when display is full
                    3 - Mark record
                    4 - A. Disable axis marks in
                            display
                        B. Disable count printout
```

To set sampling rate type R then two hex numbers (0-F) the first number sets the base rate.

A = 1/hr
9 = 1/min
7 = 1/10 sec
6 = 1/sec
5 = 10/sec
4 = 100/sec
3 = 1000/sec
2 = 10,000/sec

The second input divides the base rate to provide a variety of sampling rates, i.e., R 4 2 = 50 per sec; R A 2 = 1 per 2 hrs.

Typing "D" causes the computer to display the contents of the display buffer. This display is made with the interrupts disabled so that the display will not change as it does during the sampling mode. The display can be cleared with sense switch 1. Calibration marks can be added using sense switch 4. The calibration marks are at .5 volt intervals on the Y axis and at points 10 locations apart along the X axis. The time reference for the X axis points depends on the sampling rate. At a rate of 1/min each axis mark would be 10 min apart. While the axis marks are displayed, the sensitivity of the X and Y inputs to the oscilloscope can be adjusted to calibrate the oscilloscope display.

Typing "A" selects the analog mode and directs the computer to make A/D conversions from A/D channel 1 at the sampling rate set. This A/D conversion data is stored at the next sequential display buffer address. Between samples the display buffer is output to the oscilloscope so that the sampled points are viewed as they are being added to the display. Sense switches 1 and 4 control the display as discussed above. Sense switch 2 halts sampling when the display buffer is full. This allows a sample & hold capability. Data can be read to the display buffer from cassette tape or saved from the display buffer onto cassette tape.

Typing "E" selects the event counting mode and directs the computer to count events between clock tics. Each event is a voltage crossing detected by the Schmitt trigger circuits and is counted by a software interrupt routine. At each clock tic, the count of the events is stored in the next display buffer location, the count register is cleared and the summing process resumed. The demonstration program samples only channel one of the three available event counting channels. The display is altered by sense switches as discussed above. Sense switch 4 allows a numerical printout of the count on the terminal at each sample time. This printout is useful only for longer duration counts because the interrupt capability is disabled during the printout and this results in counting errors.

Typing "R" places the computer in the rate setting mode. Data sampling rates are variable between one in several hours and a maximum rate of 3000 analog samples per second or 17,000 event counts per second. The rate is set by typing R and a number to set the clock timer to a base rate. The base rate codes vary between "A" for one sample/hour and "2" which gives a rate of 10,000/sec (see above). The base rate is divided into sub rates by typing a second number after the R selection. As an example R 3 2 would select a base rate of 1000/sec (3) and then divide this rate by the second number 1000/2 = 500/sec. The range of divisors is limited to a single hexadecimal digit. (1 thru F equivalent to decimal numbers 1 thru 15). Thus R 3 F would equal 1000/15 = 66/sec. This division process allows a finer gradation in sampling rates.

Typing "B" causes the computer to transfer control to the JPC Monitor which we use for our tape control. This jump can be changed to transfer control to MIKBUG by changing locations 061C and 061D from 7000 to E0E3).

INTERFACE PORT ASSIGNMENTS

Port	Address	Program Label	Bits	Function
Clock Board (MP-T)				
5A	8014	SNS	0-3	A/D
			4-7	Sense Sw.
	8015	SNS+1	CA1	Detector 1
			CA2	Relay 1
5B	8016	CLOCK		Clock Func.
Parallel Port (MP-L)				
6A	8018	Y	0-7	D/A (vertical out)
	8019	Y+1	CA1	Detector 2
			CA2	Relay 2
6B	801A		0-7	Available
	801B		CB1	Available
		BRIT	CB2	CRO Intensity
Parallel Port (MP-L)				
7A*	801C	B	0-7	D/A (horizontal out)
			CA1	Detector 3
	801D	B+1	CA2	Relay 3
7B	801E-F			Available

*Port 7A is also used for line printer output when not used for D/A conversion.

PARTS LIST

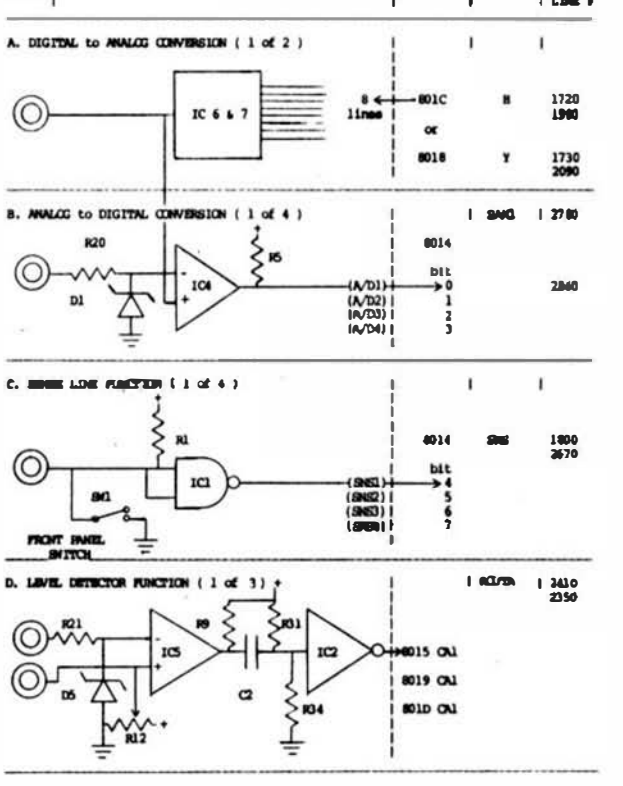
1 IC1	7400	NAND Gate	Sense Line
1 IC2	7404	Hex inverter	Monostable
1 IC3	7407	Hex Driver	Relay Drivers & Blanking
2 IC4-5	LM339	Quad Comparator	Detectors, A/D
2 IC6-7	DAC-49	Digital-to-Analog Modules	D/A & A/D
All resistors 1/4 watt			
4 R1-4	2.2K		Sense Switch
7 R5-8	3.3K		A/D Output,
3 R9-11	10K		Monostable
3 R12-14	30K	Variable Pot.	Level Detector Ref. Voltages
1 R15	10		
1 R16	470		
1 R17	4.7K		
1 R18	3.9K		Blanking
8 R20-27	100K		A/D Input
3 R28-30	100		Relay
3 R31-33	1K		Monostable
3 R34-36	2.2K		Monostable
1 C1	.01 mFd.		Despiking Cap.
3 C2-4	1 mFd.12 v		Monostable
7 D1-07	IN751 5.1 v Zener		Input protect
3 D8-10	IN914 or equivalent		Back current Protection
7 SW1-7	SPDT		Sense lines
1 SW8	11 pos. Rotary Switch		
3 Relay	Potter Brumfield 12v Relay (10 watt contacts)		

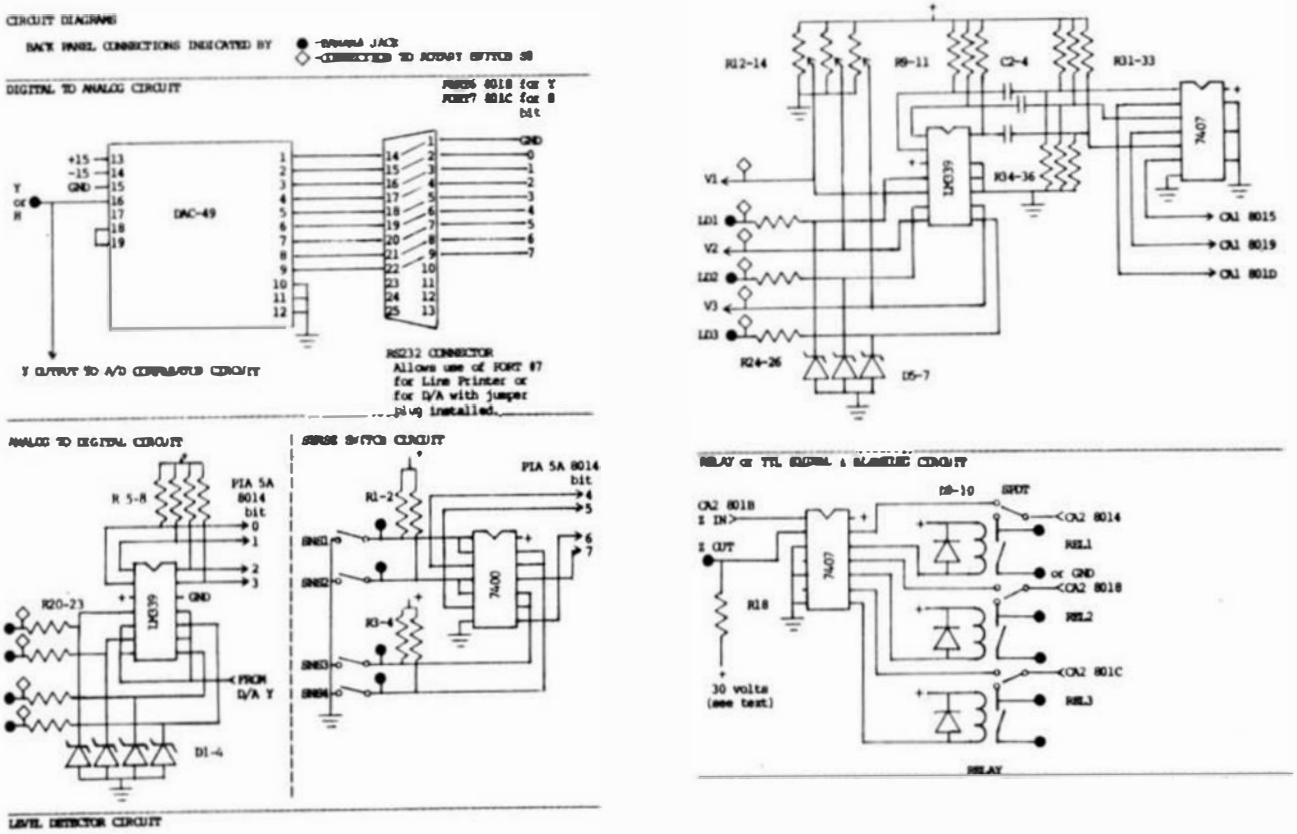
```
00010                   NAM     extended INTERFACE FOR THE 6800
00020                   OPT     O,NOMEM,NOPAGE,NOCC
00030                   *       L. ALLEN MICHAEL, OHIO STATE UNIV.
00040                   * VERSION 7/9/84
00050      A000         A000    EQU     $A000           /INTERRUPT VECTOR
00060      A048         A048    EQU     $A048           /BRANT VECTOR
00070                   * TYPE R IN MIKBUG TO DUMP DATA BUFFER TO TAPE
00080                   * TYPE L IN MIKBUG TO LOAD DATA BUFFER FROM TAPE
00090                   * USE A002 $A00A USED BY P I L FOR TAPE OPERATIONS
00100      A002         STSTR   EQU     $A002           /  START ADDRESS
00110      A004         ENDSTR  EQU     $A004           /  END ADDRESS
00120      000A         CR      EQU     $000A           / LINE FEED AND CARRIAGE RETU
00130      E07E         PDATA1  EQU     $E07E           / MIKBUG OUTPUT SUBROUTINE
00140      E1AC         INEEE   EQU     $E1AC           /KEYBOARD INPUT VIA MIKBUG
00150      7000         MON     EQU     $7000           /JPC MONITOR ADDRESS
00160      8014         SNS     EQU     $8014           /SENSE SWITCH & A-D PORT
00170      8015         REL1    EQU     $8015           /TTL OR RELAY ADDRESS
00180      8016         CLOCK   EQU     $8016           /CLOCK PORT
00190      8018         Y       EQU     $8018           /VERTICLE SIGNAL TO CRO
00200      8019         BRIT    EQU     $8019           /USE CA2 FOR CRO BRIGHTNESS
00210      801C         B       EQU     $801C           /HORIZONTAL SIGNAL TO CRO

00230 00D0                         ORG     $00D0
00240 00D0 0001          DCOUNT  RMB     1               /DISPLAY COUNTER
00250 00D1 0002          FQ1     RMB     2               /A-D CONVERT COUNT
00260 00D3 0001          TEMP    RMB     1               /TEMPORARY STORAGE LOC.
00270 00D4 0001          DIGIT   RMB     1               /TEMP. STORE FOR DECIMAL #'S
00280 00D5 0001          MF      RMB     1               /FLAG FOR MARK ROUTINE
00290 00D6 0001          BIN     RMB     1               /FLAG TO SELECT COUNT OR A/D
00300 00D7 0001          DIV     RMB     1               /MAIN DIVISOR
00310 00D8 0001          DI      RMB     1               /TEMP RATE DIVISOR
00320 00D9 0002          RADDR   RMB     2               /DISPLAY POINTER
00330 00DB 0001          HVAL    RMB     1               /TEMP. H VALUE
00340 00DC 0001          YVAL    RMB     1               /TEMP. Y VALUE
00370 00DD FF9C          CON     FDB     -100            /RAM 9 FOR BINARY-DECIMAL
00380 00DF FFF6                  FDB     -10             /  CONVERSION
00390 00E1 FFFF                  FDB     -1
```

```
00400 00E3 0002     STA     RMB     2       /STORAGE POINTER
00410 00E5 0D       STORE   FCB     $01,$31,$00,$32,$00,$33,$0D
00420 00EC 30       SAMP    FCB     $30,$10 /SAMPLE 0
00430 00EE 0000             FDB     CR
00440 00F0 04               FCB     4



00460 00FF                  ORG     $FF     / INITIALIZATION
00470 00FF 3F               SWI             / FALSE START TO RESET POINTE
00480 0100 0F       BEGIN   SEI             / SET INTERUPT MASK
00490 0101 BE A042          LDS     #$A042  /PROTECT STACK
00500 0104 CE 02B2          LDX     #INSERV /GET ADDR. OF INTERUPT SERVIC
00510 0107 FF A000          STX     $A000   /  AND STORE FOR INTERUPT VEC
00520 010A CE 0100          LDX     #BEGIN  /GET ADDR. OF PRO . BEGINNING
00530 010D FF A046          STX     $A046   /  AND SAVE FOR MINIBUG START
00540 0110 7F 00D5          CLR     MF      /CL AR MARK FLAG
00550 0113 7F 00D1          CLR     POT     /CLEAR COUNTER LOC.
00560 0116 7F 00D2          CLR     POT+1
00570 0119 7F 00D4          CLR     DIGIT   /CLEAR TEMP STORAGE LOC.
00580 011C CE 0B00          LDX     #$0B00  /GET & SAVE STARTING
00590 011F DF E3            STX     STR     /  LOC. OF DATA STORAGE

00600 0121 FF A002          STX     $1919   /SAVE BUFF.ADDR. FOR TAPE OP.
00610 0124 CE 0A00          LDX     #$0A00  /GET END ADDRESS OF STORAGE
00620 0127 FF A004          STX     ENSTR   /  AND SAVE FOR TAPE OPERATIO
00630 012A B6 30            LDA A   $0030   /30-ASCII ZERO
00640 012C 97 EC            STA A   SAMP    /SET SAMPLE NO. TO ZERO
00650 012E 97 ED            STA A   SAMP+1
00660 0130 7F 8010          CLR     M+1     /CLEAR CONTROL REGISTERS
00670 0133 7F 8019          CLR     Y+1
00680 0136 7F 8017          CLR     CLOCK+1
00690 0139 7F 8015          CLR     SNS+1
00700 013C 7F 8014          CLR     SNS     /S 1 UP AS ALL INPUTS
00710 013F B6 FF            LDA A   $FF
00720 0141 B7 8018          STA A   M       / SET ALL OUTPUTS
00730 0144 B7 8018          STA A   Y       /   "
00740 0147 B7 8016          STA A   CLOCK   /   "
00750 014A B6 04            LDA A   $04
00760 014C B7 801D          STA A   M+1     /TURN OFF DDR
00770 014F B7 8019          STA A   Y+1     /   "
00780 0152 B6 3D            LDA A   $3D
00790 0154 B7 8015          STA A   SNS+1   / E AND INTERUPT ON CA1
00800 0157 B7 8017          STA A   CLOCK+1 /SET UP CLOCK
00810 015A CE 060A          LDX     #MESS4  /LOAD ADDRESS OF MESSAGE 4
00820 015D BD E07E          JSR     PDATA1  /JMP TO PRINT OUT ROUTINE
00830 0160 CE 0606 KBIN     LDX     #MESS3  /DISPLAY ":" PROMPT
00840 0163 BD E07E          JSR     PDATA1
00850 0166 BD E1AC          JSR     INEE    /KEYBOARD INPUT SELECTION
00860 0169 81 4D            CMP A   #$4D    /M 0010 MONITOR
00870 016B 26 03            BNE     B1
00880 016D 7E 7000          JMP     MON
00890 0170 81 52    B1      CMP A   #$52    /R SET CLOCK RATE
00900 0172 26 03            BNE     B2      /NOT R SKIP PAST
00910 0174 7E 026D          JMP     TSET    /DO SET CLOCK RATE
00920 0177 81 41    B2      CMP A   #$41    /A A/D MODE
00930 0179 26 03            BNE     B3      / NOT A/D MODE
00940 017B 7C 00D6          INC     FLG     /SET FLAG-SEE ROUTB
00950 017E 81 45    B3      CMP A   #$45    /E EVENT COUNTING MODE
00960 0180 26 03            BNE     B4      /NOT E SKIP PAST
00970 0182 7F 00D6          CLR     FLG     /CLEAR FLG-SEE ROUTB
00980 0185 81 44    B4      CMP A   #$44    /D DISPLAY STORAGE BUFFER
00990 0187 26 02            BNE     B5      /NOT D SKIP PAST
01000 0189 20 08            BRA     LOOP    /GOTO LOOP INTERUPTS OFF
01010 018B 81 49    B5      CMP A   #$49    /I DISPLAY INFORMATION
01020 018D 26 02            BNE     INTON   / NO SKIP FAST
01030 018F 20 30            BRA     CMD
01040 0191 01       INTON   NOP             / NOP REQ. BEFORE CLI
01050 0192 0E               CLI             /ENABLE INTERUPTS
01060 0193 B6 8014 LOOP     LDA A   SNS     /CHECK EXTERNAL SENSE LINES
01070 0196 84 70            AND A   #$70    /MASK OUT A/D FLAGS
01080 0198 81 10            CMP A   #$10    /** IS SNS 1 ON
01090 019A 27 0A            BEQ     CLEAR   /  YES, CLEAR STORAGE & RESTA
01100 019C 81 40            CMP A   #$40    /** IS SNS 3 ON
01110 019E 27 30            BEQ     MARK    /  YES, MARK RECORD
01120 01A0 BD 01F7          JSR     DISP    /DISPLAY STORAGE LOC'S.
01130 01A3 7E 0193          JMP     LOOP    /WAIT IN LOOP FOR INTERUPTS


01150               ********SUBROUTINES********

01170               * SUBROUTINE TO CLEAR STORAGE AND RESTART
01180 01A6 CE 0B00  CLEAR   LDX     #$0B00  /SET STORAGE POINTER (STR)
01190 01A9 DF E3            STX     STR     /  TO BEGINING OF STORAGE
01200 01AB 6F 00    CLST    CLR     X       / CLEAR DISPLAY STORAGE
01210 01AD 8C 0A00          CPX     #$0A00  /  (B00-A00 HEX) 256 POINTS
01220 01B0 27 03            BEQ     CLS1    / BRANCH WHEN FINISHED
01230 01B2 08               INX             / INCREMENT TO NEXT LOC.
01240 01B3 20 F6            BRA     CLST    / CLEAR NEXT LOCATION
01250 01B5 7F 00D5  CLS1    CLR     MF      /CLEAR MARK FLAG
01260 01B8 86 30            LDA A   $0030   /LOAD ASCII ZERO
01270 01BA 97 EC            STA A   SAMP    /SET SAMPLE NO. TO ZERO
01280 01BC 97 ED            STA A   SAMP+1
01290 01BE 7E 0191          JMP     INTON   / GO TO SAMPLING CONFIDERATIO

01310               *SUBPROGRAM TO DISPLAY INPUT INFO
01320 01C1 CE 049B  CMD     LDX     #MESS2
01330 01C4 BD E07E          JSR     PDATA1
01340 01C7 CE 036F          LDX     #MESS1
01350 01CA BD E07E          JSR     PDATA1
01360 01CD 7E 0160          JMP     KBIN    /RETURN FOR KEYBOARD INPUT

01380               *SUBPROGRAM TO PUT MARK INTO D TA STRING
01390 01D0 4F       MARK    CLR A           /SET A=0 AND CHECK MF=0
01400 01D1 91 D5            CMP A   MF      /CK. FLG - ALLOWS ONLY ONE
01410 01D3 27 03            BEQ     M1      /   MARK PER RECORD
01420 01D5 7E 0193          JMP     LOOP    /RETURN IF ALREADY MARKED
01430 01D8 DE E3    M1      LDX     STR     / GET STORAGE ADDRESS
01440 01DA 86 00            LDA A   #$00    / PUT MCX CODE INTO DATA STRI
01450 01DC C6 FE            LDB B   #$FE    / MARK IS AT TOP OF DISPLAY
01460 01DE A7 00            STA A   0,X     /MAKE THREE MARKER POINTS
01470 01E0 E7 01            STA B   1,X
01480 01E2 A7 02            STA A   2,X
01490 01E4 E7 03            STA B   3,X
01500 01E6 A7 04            STA A   4,X
01510 01E8 E7 05            STA B   5,X
01520 01EA 08               INX             /UPDATE STORAGE POINTER 6X
01530 01EB 08               INX
01540 01EC 08               INX
01550 01ED 08               INX
01560 01EE 08               INX
01570 01EF DF E3            STX     STR     /SAVE NEW X ADDRESS
01580 01F1 7C 00D5          INC     MF      /SET FLG -ONE MARK/RECORD
01590 01F4 7E 0193          JMP     LOOP    /RETURN

01610               * SUBROUTINE TO DISPLAY DATA
01620 01F7 7F 00DB  DISP    CLR     HVAL    /CLEAR TEMP. H VALUE
01630 01FA CE 0B00          LDX     #$0B00  /DISPLAY LOC. B00 (1ST)
01640 01FD DF D9            STX     XADDR   /POINTER TO START OF STORAGE
01650 01FF DE D9    DISP1   LDX     XADDR   /LOAD POINTER
01660 0201 E6 03            LDA B   3,X     /SAVE DATA POINT #1
01670 0203 D7 D3            STA B   TEMP    / FOR RETURN FROM INTERUPT
01680 0205 E6 01            LDA B   1,X     /LOAD DATA FROM STORAGE
01690 0207 C1 00            CMP B   #$00    /DON'T DISPLAY ZERO'S
01700 0209 27 0B            BEQ     ON
01710 020B 96 DB            LDA A   HVAL    /LOAD TEMP HORZ. VAL
01720 020D B7 801C          STA A   H       /PUT TIME INTO HORZ. DISPLAY
01730 0210 F7 8018          STA B   Y       /LOAD DATA POINT INTO PORT
01740 0213 BD 0262          JSR     BRIGHT  /INTENSIFY POINT
01750 0216 08               INX             /INCERMENT STORAGE POINTER
01760 0217 08       ON      INX             /  "
01770 0218 DF D9            STX     XADDR   /SAVE POINTER
01780 021A 7C 00D8          INC     HVAL    /HVAL HORZ. VALUE ONE LOC.
01790 021D 26 E0            BNE     DISP1   /DISPLAY NEXT POINT
01800 021F 96 8014          LDA A   SNS
01810 0222 84 80            AND A   #$80    /CHECK SNS B4 ON
01820 0224 27 01            BEQ     CALC    /  YES-DISPLAY AXIS MARKS
01830 0226 39               RTS
01840 0227 BD 022B  CALC    JSR     CAL
01850 022A 39               RTS             /RETURN

01870               *SUBROUTINE TO CALIBRATE OSCILLOSCOPE
01880 022B 7F 00D8  CAL     CLR     Y       /MARK X & Y AT 50 INTERVALS
01890 022E 7F 00D9          CLR     HVAL    /SET X VALUE TO ZERO

01900 0231 7F 001C          CLR     H
01910 0234 BD 0262          JSR     BRIGHT  /INTENSIFY POINT AT 0,0
01920 0237 8A 0A            LDA A   #10
01930 0239 97 DB            STA A   HVAL    /PUT 10 INCERMENT INTO HTEMP
01940 023B 86 33            LDA A   #51
01950 023D 97 DC            STA A   VVAL    /PUT 1 VOLT MARKS INTO YTEMP
01960 023F 4F               CLR A
01970 0240 9B DB    HLOOP   ADD A   HVAL    /XAXIS MARKS 10 APART
01980 0242 25 0B            BCS     H1      /BRANCH WHEN FINISHED
01990 0244 B7 801C          STA A   H       /PUT 10,0-20,0-30,0 ETC AXIS
02000 0247 BD 0262          JSR     BRIGHT  /DISPLAY X AXIS POINTS
02010 024A 20 F4            BRA     HLOOP   /DO NEXT POINT
02020 024C 7F 801C  H1      CLR     H
02030 024F 86 A0            LDA A   #$A0    /WAIT FOR X TO STABALIZE
02040 0251 4A       WAITX   DEC A
02050 0252 26 FD            BNE     WAITX   /   "
02060 0254 4F               CLR A
02070 0255 9B DC    VLOOP   ADD A   VVAL    /Y AXIS MARKS 51 APART
02080 0257 25 0B            BCS     LRTN    /BRANCH WHEN FINISHED
02090 0259 B7 801B          STA A   Y       /PUT 0,51-0,102-0,153 ETC TAB
02100 025C BD 0262          JSR     BRIGHT  /DISPLAY Y POINT
02110 025F 20 F4            BRA     VLOOP   /GET NEXT POINT
02120 0261 39       LRTN    RTS             /RETURN

02140               * SUBROUTINE TO INTENSIFY CRO POINT
02150 0262 C6 34    BRIGHT  LDA B   #$34    /CA2 ON--910MALS
02160 0264 F7 801B          STA B   PRB     / DON'T INTENSITY ON
02170 0267 C6 3C            LDA B   #$3C
02180 0269 F7 801B          STA B   PRB     / Z OFF
02190 026C 39               RTS             /RETURN

02210               *SUBROUTINE TO INPUT BAUD INFO.
02220 026D BD E1AC  TSET    JSR     INEE    /GET DESIRED RATE FROM KEYBOA
02230 0270 B7 8016          STA A   CLOCK   /PUT RATE CODE IN CLOCK
02240 0273 BD E1AC          JSR     INEE    /GET RATE DIVIDE FACTOR
02250 0276 84 0F            AND A   #$0F    /MASK A/D LIT HALF
02260 0278 26 01            BNE     BRD     /CK. IF ZERO
02270 027A 4C               INC A           /  YES DEFAULT TO 1
02280 027B 97 DB    BRD     STA A   DI      /SAVE RATE DIVISOR
02290 027D 97 D7            STA A   DIV     /   "   "
02300 027F 7E 0160          JMP     KBIN    /RETURN

02320               *INTERUPT SERVICE ROUTINE

02340 0282 F6 8015  INSERV  LDA B   SNS+1   /CA SCHMITT TRIGGER INTERUPT
02350 0285 2B 06            BMI     ROUTA   /EVENT INTERUPT
02360 0287 F6 8017          LDA B   CLOCK+1
02370 028A 2B 0D            BMI     ROUTB   /CLOCK INTERUPT
02380 028C 3B               RTI

02400               *COUNT EVENTS ROUTINE
02410 028D F6 8014  ROUTA   LDA B   SNS     /CLEAR PIA
02420 0290 7C 00D2          INC     POT+1   /INCERMENT COUNTER(LOW ORDER)
02430 0293 26 03            BNE     RTN     / IS THERE A CARRY **
02440 0295 7C 00D1          INC     POT     /   YES-INCERMENT HIGH ORDER
02450 0298 3B       RTN     RTI             / RETURN

02470               *SERVICE CLOCK INTERUPT ROUTINE
02480 0299 F6 8016  ROUTB   LDA B   CLOCK   /CLEAR PIA
02490 029C 7A 00DB          DEC     DI      /WAIT DIVISOR CHECK
02500 029F 26 19            BNE     RETN
02510 02A1 96 D7            LDA A   DIV
02520 02A3 97 DB            STA A   DI      /UPDATE DIVISOR
02530 02A5 5F               CLR B

02560 02A6 D1 D4            CMP B   FLG     /CK FLAG - IF ZERO
02550 02A8 27 50            BEQ     COUNT   /  GOTO COUNTER MODE
02560 02AA BD 02DB          JSR     SAM1    /GET A-D VALUE
02570 02AD DE E3            LDX     STR     /LOAD STORAGE ADDRESS
02580 02AF A7 01            STA A   1,X     /PUT DATA POINT IN STORAGE
02590 02B1 08               INX             /INCERMENT STORAGE POINTER(1ST
02600 02B2 08               INX             /   "   2 TIMES
02610 02B3 DF E3            STX     STR     /UPDATE STORAGE ADDRESS
02620 02B5 8C 0A00          CPX     #$0A00  /CHECK FOR END OF STORAGE
02630 02B8 27 01            BEQ     FULL    /   YES-GOTO FULL
02640 02BA 3B       RETN    RTI             /   NO-RETURN
02650 02BB CE 0B00  FULL    LDX     #$0B00  /RESET STORAGE POINTER
02660 02BE DF E3            STX     STR
02670 02C0 96 8014  FULL1   LDA A   SNS     /IF SNS 2 IS ON HOLD DISPLAY
02680 02C3 84 20            AND A   #$20    / WITH INTERUPTS OFF
02690 02C5 26 06            BNE     RNT
02700 02C7 CA 34            LDA B   #$34
02710 02C9 F7 8015          STA B   RELI    /TTL OR RELAY OFF
02720 02CC 3B               RTI             /RETURN -INTERUPTS ON
02730 02CD BD 01F7  RNT     JSR     DISP    /DISPLAY DATA
02740 02D0 C6 3C            LDA B   #$3C
02750 02D2 F7 8015          STA B   RELI    /TTL LEVEL OR RELAY ON
02760 02D5 7E 02C0          JMP     FULL1   /LOOP TO DISPLAY DATA

02780 02D8 4F       SAM1    CLR A           /A-D SAMPLING ROUTINE
02790 02D9 C6 B0            LDA B   #$B0    /ROTATING MASK
02800 02DB 1B       BITLOP  ABA             /ADD A & B
02810 02DC B7 B010          STA A   Y       /SEND A+B TO COMPARATOR
02820 02DF 01               NOP             /WAIT FOR B-A TO SETTLE
```

```
02830 02E0 01                NOP             /   *
02840 02E1 B6 8014           LDA A  SNS      / CHECK COMPARATOR OUTPUT
02850 02E4 84 01             AND A  #$01     /IS COMP. 01 ON*?
02860 02E6 27 06             BEQ    SAVE     / NOT YET. SAVE BIT
02870 02E8 B6 801F           LDA A  Y        / YES.ESTIMATE IS HIGH
02880 02EB 10                SBA             / SUB. BIT MASK FROM ESTIM.
02890 02EC 20 03             BRA    ROTB     /SKIP PAST SAVE
02900 02EE B6 8018 SAVE      LDA A  Y        /KEEP CURRENT ESTIMATE
02910 02F1 56        ROTB    ROR B           /ROTATE BIT MASK
02920 02F2 24 E7             BCC    BITLOP   /LOOP UNTIL Y=UNKNOWN
02930 02F4 D6 D3             LDA B  TEMP     /LOAD Y&B WITH DATA TO CORREC
02940 02F6 F7 8018           STA B  Y        /INTER.NFB.DISP.ERR.
02950 02F9 39                RTS             /CONVERSION FINISHED.RETURN

02970 02FA 96 D1   COUNT     LDA A  POI      /COUNTER MODE ROUTINE
02980 02FC D6 D2             LDA B  POI+1    /PUT COUNT INTO POT
02990 02FE DE E3             LDX    STR      /LOAD STORAGE LOC.
03000 0300 A7 00             STA A  0,X      /STORE 2 BYTE COUNT
0 010 0 02 E7 01             STA B  1,X
03020 0304 08                INX             /INC POINTER
03030 0305 08                INX             /   *   2X
03040 0306 DF E3             STX    STR      /SAVE NEW STORAGE LOC.
03050 0308 8C 0A00           CPX    #$0A00   /CHECK FOR STORAGE FULL
03060 030B 27 AE             BEQ    FULL     /   YES  GOTO FULL
03070 030D CE 00CC           LDX    #BAMP    /INDEX SAMPLE COUNTER
03080 0310 86 39             LDA A  #$39     /ASCII 9
03090 0312 C6 30             LDA B  #$30     /ASCII ZERO
03100 0314 A1 01             CMP A  1,X      /IS 1ST DIGIT=9
03110 0316 27 04             BEQ    T1       /   YES, INC. TO 0 & CARRY
03120 0318 6C 01             INC    1,X      /   NO INC. & DISPLAY
03130 031A 20 0E             BRA    NEXT3
03140 031C E7 01   T1        STA B  1,X
03150 031E A1 00             CMP A  0,X      /*? IS 2ND DIGIT *9
03160 0320 27 04             BEQ    NEXT1    /   YES, SET IT TO ZERO
03170 0322 6C 00             INC    0,X      /   NO INCERMENT 2ND DIGIT
03180 0324 20 02             BRA    NEXT2    /SET 1ST DIG. TO ZERO
03190 0326 E7 00   NEXT1     STA B  0,X      /PUT ZERO INTO 2ND DIGIT

03200 0328 E7 01   NEXT2     STA B  1,X      /PUT ZERO INTO 1ST DIGIT
03210 032A B6 8014 NEXT3     LDA A  SNS      /IF SW8 2=1 SKIP
03220 032D 84 80             AND A  #$80     / VIDIO READOUT
03230 032F 27 03             BEQ    BDCONV   / OF COUNT
03240 0331 7E 0368           JMP    NEXT4

03260 0334 86 03   BDCONV    LDA A  #03      /BINARY TO DECIMAL CONVERSION
03270 0336 97 D0             STA A  DCOUNT   / COUNT UP TO 3 DIGITS
03280 0338 7F 00D4           CLR    DIGIT    / INITILIZE TO ZERO
03290 033B CE 00DD           LDX    #COM     / PUT 16 BIT DECIMAL MASK
03300 033E 96 D1   SUBTR     LDA A  POT      /   INTO X
03310 0340 D6 D2             LDA B  POT+1    / LOAD A & B WITH COUNT
03320 0342 EB 01             ADD B  1,X      / SUBTRACT MASK FROM COUNT
03330 0344 A9 00             ADC A  0,X
03340 0346 2D 0A             BLT    SAVDI    / IS THE RESULT NEGATIVE
03350 0348 7C 00D4           INC    DIGIT    /   NO-ADD ONE TO DIGIT
03360 034B 97 D1             STA A  POT      / UPDATE COUNTER
03370 034D D7 D2             STA B  POT+1
03380 034F 7E 033E           JMP    SUBTR
03390 0352 96 D4   SAVDI     LDA A  DIGIT    / SAVE CURRENT DIGIT
03400 0354 8A 30             ORA A  #$30     / CONV DIGIT TO ASCII
03410 0356 A7 09             STA A  9,X      / AND STORE IN  ATA STRING
03420 0358 7F 00D4           CLR    DIGIT
03430 035B 08                INX             / INCERMENT TO GET NEW
03440 035C 08                INX             /   DECIMAL MASK
03450 035D 7A 00D0           DEC    DCOUNT   / DOES THIS MAKE 3 DIGITS
03460 036  26 DC             BNE    SUBTR    /   NO- GET NEXT DIGIT

03480 0362 CE 00E5           LDX    #STORE   / DISPLAY COUNT & TIME
03490 0365 BD E07E           JSR    PDATA1   /OUTPUT ROUTINE
03500 0368 7F 00D1 NEXT4     CLR    POT
03510 036B 7F 00D2           CLR    POT+1
03520 036E 3B                RTI             / RETURN TO LOOP

03540 036F 20        MES01   FCC    /  TO SET SAMPLING RATE TYPE R THEN/
03550 0391 0D0A              FDB    CR
03560 0393 20                FCC    /      ONE KEY TO SET BASE RATE./
03570 03B2 0D0A              FDB    CR
03580 03B4 0D0A              FDB    CR
03590 03B6 20                FCC    26,    A =1/HR       5 =10/SEC
03600 03D0 0D0A              FDB    CR
03610 03D2 20                FCC    27,    9 =1/MIN      4 =100/SEC
03620 03ED 0D0A              FDB    CR
03630 03EF 20                FCC    28,    7 =1/10SEC    3 =1000/SEC
03640 040B 0D0A              FDB    CR
03650 040D 20                FCC    29,    6 =1/S C      2 =10000/SEC
03660 042A 0D0A              FDB    CR
03670 042C 0D0A              FDB    CR
03680 042E 20                FCC    / THEN TYPE ONE 0 TO DIVIDE BASE RATE
03690 0452 0D0A              FDB    CR
03700 0454 20                FCC    /      I.E. R 4 2 = 50 PER SEC/
03710 0471 0D0A              FDB    CR
03720 0473 20                FCC    /              R A 2 = 1 PER 2 HRS./
03730 0495 0D0A              FDB    CR
03740 0497 04                FCB    4
03750 0498 20        MES02   FCC    /  AT (:) ENTER A,E,D,R OR R+B'S/
03760 04B7 0D0A              FDB    CR
03770 04B9 20                FCC    *    A- A/D SAMPLE MODE*
03780 04CF 0D0A              FDB    CR
03790 04D1 20                FCC    /    E- EVENT COUNTING MODE/
03800 04EB 0D0A              FDB    CR
03810 04ED 20                FCC    /    D- DISPLAY STORAGE & CALIBRATION
03820 0511 0D0A              FDB    CR
03830 0513 20                FCC    /    M- GOTO JPC MONITOR/
03840 052A 0D0A              FDB    CR
03850 052C 20                FCC    /    R- SET SAMPLING RATE/
03860 0544 0D0A              FDB    CR
03870 0546 0D0A              FDB    CR
03880 0548 0D0A              FDB    CR
03890 054A 53                FCC    22,SENSE SWITCH FUNCTION
03900 0560 0D0A              FDB    CR
03910 0562 20                FCC    32,  1  -CLEAR & RESTART
03920 0582 0D0A              FDB    CR
03930 0584 20                FCC    /  2  -HOLD WHEN N DISPLAY IS FULL./
03940 05A4 0D0A              FDB    CR
03950 05A6 20                FCC    20,  3  -MARK RECORD
03960 05BA 0D0A              FDB    CR
03970 05BC 20                FCC    /  4 A. DISABLE AXIS MARKS IN DISPLAY
03980 05E0 0D0A              FDB    CR
03990 05E2 20                FCC    /    B. DISABLE COUNT PRINTOUT/
04000 05FF 0D0A              FDB    CR
04010 0601 0D0A              FDB    CR
04020 0603 0D0A              FDB    CR
04030 0605 04                FCB    4
04040 0606 0D0A      MES03   FDB    CR
04050 0608 3A                FCC    /:/
04060 0609 04                FCB    4
04070 060A 0D0A      MES04   FDB    CR
04080 060C 20                FCC    / COMMANDS: A E D R M  FOR INFO./
04090 062B 0D0A              FDB    CR
04100 062F 04                FCB    4
04110                        END

TOTAL ERRORS 00000

ENTER PASS # 1P,2P,2L,2T
```

FIGURE 1





FLEX SPOOLER PRINTER CONTROL

by Barry Ballitski
151 Midglen Place
Calgary, Alberta
Canada T2X 1H6

I am the proud owner of a 6809 FLEX system which was built up over many years from different pieces of equipment. The printer I have is an antique G.E. TERMINET 300, this printer uses a rotating belt which has all the characters on it and the correct letter is hit by one of the 118 hammers putting the impression on the paper. This arrangement is noisy even when running without printing text as the belt always rotates when the machine is selected. The other major aggravation is that if you try to tear off the exposed paper when the belt is running you risk snagging the paper on the print fingers which shreds the paper and quite frequently deforms the print fingers, requiring a steady hand and much patience and testing to reform to the correct position. This arrangement had me getting up from my chair to select the printer to 'on-line', typing the command, and having to get up to de-select the printer when done printing. This promoted me to dig into FLEX to see if there was some way I could control the printer on and off functions automatically when doing printer spooling. After a bit of research and experimentation I found it could be done quite easily. The following article will outline how I implemented it on my system and on a GIMIX system. The main differences between the two are dependant on whether you have the general version of FLEX (which you have the I/O routines for) or an already configured system such as GIMIX, SWTPc or similar. I found in most systems which are already configured there is an obvious lack of documentation on the I/O, interrupt and disk routines There is a manual available from TSC (developers of FLEX) called the 'FLEX 6809 Adaptation Guide' this manual is $25.00 and worth it's weight in gold. This manual is included in only the 'FLEX for general use' and is not included in GIMIX or SWTPc FLEX versions. The adaptation guide includes information on such things as : Console I/O driver package, Disk driver package, developing and testing disk drivers, bootstrap loading of FLEX, NEWDISK routine, Printer spooling, interrupt handling, advanced disk adaptation, additional customization and other valuable information for anyone who wants to adapt FLEX or just know how it works.

These routines may be used not only to control printer on/off but also to beep, form feed, multiple line feed or whatever desired.

Adapting to my system

Because of my having to write my own I/O routines it was very easy to change my printer initialization at $CCCO (everybody's is in the same location) to include the sending out of an escape character and an H character, this turns the printer on. This modified code will start my printer whether I use the P.CMD or the PRINT.CMD as they both use the same printer initialization. The modified code for my printer initialization is as follows :

```
PRACIA  EQU     $E000       PRINTER ACIA
POUT    EQU     $CCE4       OUTPUT TO PRINTER

        ORG     $CCCO       INITALIZE PRINTER
PINIT   LDA     #$13        RESET THE ACIA
        STA     PRACIA
        LDA     #$11        8 BITS, 2 STOP
        STA     PRACIA
*=======================================!
*    This is all the code I had to add  !
ON      LDA     #$1B        ESCAPE CHAR    !
        BSR     POUT        TO PRINTER     !
        LDA     #$48        'H' CHAR       !
        BSR     POUT        PRINTER IS ON  !
*=======================================
        RTS                 DONE INITALIZATION
```

******** WARNING ********

Code can't extend past $CCD7 as the PCHK (printer check) routine starts at $CCD8. Don't over write that code or .....CRASH !!

Adapting FLEX to turn off the printer was also an easy task for me as it only required I modify the TMOFF (interrupt timer off) routine to check to see if the timer was being shut off with no files left in the queue. This would indicete the printer could be shut off. My printer requires an 'escape' char. and a 'J' char. to de-select. The modified TMOFF routine is as follows :

```
QCNT    EQU     $C71B       QUEUE COUNT
POUT    EQU     $CCE4       OUTPUT TO PRINTER

TMOFF   LDA     #$8P        TIMER OFF PATTERN
        STA     $E012       MY INTERRUPT TIMER

*=======================================
* code to shut off printer after spooling !
        LDA     QCNT        TEST QUEUE COUNT !
        BNE     BACK        STILL SPOOLING ? !
        LDA     #$1B        NO               !
        JSR     POUT        SEND ESCAPE      !
        LDA     #$4A        'J' CHAR.        !
        JSR     POUT        PRINTER IS OFF   !
*=======================================
BACK    RTS
```

Both of these routines have been in use for several months and no problems have been detected, nor do I think any other FLEX systems will have any trouble with an implementation of this type.

Adapting to GIMIX FLEX 4.3

Because you (unfortunately) don't get the source code for the printer or console routines, much of the code had to be disassembled to understand it's operation enough to modify it for printer control. The printer used here requires a $11 to select or $13 to de-select. The printer initalize code was disassembled and room was found at the end of PINIT code for the select printer function which follows.

(Assemble following code and GET,ONCODE.BIN)

```
**   GIMIX SYSTEM ONLY   **
**   PARALLEL PRINTER    **

CONREG  EQU     $E043       PIA CONTROL REG
DATREG  EQU     $E042       PIA DATA REG.

POUT    EQU     $CCE4       PRINTER OUTPUT

        ORG     $CCCO       ALL FLEX VERSIONS

PINIT   CLR     CONREG       DIRECTION REG
        LDA     #$FF        ALL LINES OUTPUT
        STA     DATREG      SET DIRECTION REG
        LDA     #$3E        CONFIGURE PATTERN
        STA     CONREG      SET CONTROL REG.
*---------------------------------------
*    Adapt to your printer requirements   !
        LDA     #$11        SELECT PRINTER !
        BSR     POUT        SEND IT OUT    !
*---------------------------------------
        RTS
```

******* WARNING *******

Don't over-write the PCHK routine at $CCD8.

After disassembling the console routines it became apparent it would not be an easy task to implement the printer off function when the source code is not available, besidea there wasn't enough room in the console area to put any extra code. This necessitated that a command be made to implement this. The command is OFFCODE.CMD which will check MEMEND, ($CC2B) lower it by the amount of space needed for the new printer off routine and place a new value in MEMEND. The original TMOFF routine vector ($D3ED) is altered to point to the new routine above MEMEND and the routine exits to the original TMOFF routine. This requires OFFCODE.CMD be executed in the utility space and it will transfer the new routine to above MEMEND and alter the vectors. The OFFCODE.CMD source follows :

```
        NAM     OFFCODE

        ORG     $C100       UTILITY SPACE
ENTRY   BRA     BEGIN
        PCB     1           VERSION NUMBER

TMOFFV  EQU     $D3ED       TMOFF VECTOR
MEMEND  EQU     $CC2B
WARMS   EQU     $CD03       WARMSTART FLEX

BEGIN   LDD     MEMEND      GET PRESENT VALUE
        SUBD    #ENDC-STARTC
        STD     MEMEND      STORE THE NEW VALUE
        LDX     TMOFFV      TIMER OFF ROUTINE
        STX     POINT       END OP NEW ROUTINE
        LDX     #STARTC     POINT X TO ROUTINE
        TFR     D,Y         Y AT NEW MEMEND
        LEAY    1,Y         NEXT FREE LOCATION
PUTONE  LDA     0,X+        GET BYTE PROM CODE
        STA     0,Y+        STORE TOP OF MEMORY
        CMPX    #ENDC       IS IT ALL DONE ?
        BNE     PUTONE      NO GO DO ANOTHER
* ALL DONE INSERTING THE NEW CODE
DONE    LDX     MEMEND
        LEAX    1,X         POINT TO FIRST CODE
        STX     TMOFFV      PUT IT IN VECTOR
        JMP     WARMS
```

```
*  FOLLOWING CODE FOR PRINTER CONTROL
*  WILL BE READ BY ABOVE CODE AND
*  TRANSFERED TO BELOW MEMEND AND THE
*  MEMEND VALUE ALTERED

QCNT    EQU     $C71B       QUEUE COUNT
POUT    EQU     $CCE4       PRINTER ROUTINE

START   EQU     *
        LDA     QCNT        IS THE QUEUE EMPTY?
        BNE     INUSE
* INSERT PRINTER DEPENDANT CODE HERE
        LDA     #$13        DE-SELECT CHAR
        JSR     POUT        PRINTER ROUTINE
INUSE   FCB     $7E         EXTENDED JUMP CODE
POINT   RMB     2           TMOFF ROUTINE
END     EQU     *

        END     ENTRY
```

The OFFCODE.CMD listed above will work with any FLEX system. Although it eats up avaliable memory it does not require you to have the source for the console or printer I/O routines.

SUMMARY

The above examples turn on the printer when using P.CMD or PRINT.CMD but the printer will not be shut off after the P.CMD. Only the use of PRINT.CMD will shut it off. This is because I could not easily see a way to do this without using non-standard FLEX routines or code. I leave this up to some other ambitious soul to crack or maybe until the next one thousand hours I have on my hands. I have found the implementation of these routines very helpful and submit them to the best (in my opinion) 68XX magazine around. Anyone requiring information or assistance please feel free to contect me.

B Bakitaki

MEMCMD.SCR

By: Ken Drexler
311 Wilson Way
Larkspur, CA 94939

MEMCMD.SRC is the source code for an assembly language program which will add three new commands to FLEX. The program uses the memory resident command feature of FLEX. See page 16, of the 6809 FLEX Programmer's Manual. The three commands are PS, FF and LC.

PS - Pause Set: This command turns on FLEX's Pause Feature after printing.

FF - Form Feed: This command sends a form feed ($0C) character to the printer which is connected to the system through POUT ($CCE4).

LC - Last Command: This command allows the user to re-run a command quickly without reloading it from disk. It operates by checking the Transfer Address in FLEX at ($CC1E) and, if it in not zero, jumps to the program at that address. If the Transfer Address is zero then the progam checks whether the byte at that address is a jump ($7E), branch ($20) or long branch ($16). If the byte at the Transfer Address is one of these, LC jumps to the program at the Transfer Address. This check is needed because FLEX sets the Transfer Address to zero if no transfer address was found. The effect of using LC is to re-run whatever command was just loaded from disk by FLEX.

All of these commands are located in memory and, as a result, are fast.

These programs require approximately 36 bytes of memory which is located above FLEX's MEMEND. RAMEND must be set equal to the location where the program is to be placed before the program is assembled. Some possible locations for the program on a FLEX 9 system are the following: Memory at $E000 or above, above the disk drivers starting at $DE00, if disk spooling function is not used, in the spooler's FCB at $CAC0 to $CBFF or, if you have General FLEX, in unused space between $D370 and $D3E5. The programs are written in source code which can be assembled for either FLEX 2 or FLEX 9 by changing only the System Equate.

After the program is assembled, it is enabled by being loaded in memory. A good way to do this is to have the STARTUP file include the command to GET MEMCMD.BIN. Once the file is loaded, the PS, FF and LC commands are enabled. These commands can be disabled only by rebooting or clearing USRCMD at $CC12.

- - -

MEMORY COMMAND PROGRAM - VER. 2.

```
                        *****MEMORY COMMAND PROGRAM*****
                        *
                        * DATE: MAY 22, 1982
                        * REVISED JULY 1, 1982 TO CHANGE ORGIN
                        * REVISED SEPTEMBER 6, 1982 TO CHANGE ORGIN
                        * REVISED NOVEMBER 1, 1982 FOR K9 VER. 2
                        * REVISED APRIL 2, 1983 TO USE POUT IN
                        * FLEX AND ALLOW PRINTERS AT DIFFERENT
                        * ADDRESSES
                        * REVISED AUGUST 26, 1983 TO ADD "LC" COMMAND
                        * REVISED SEPTEMBER 3, 1984 TO ALLOW PROGRAMS
                        * BE RUN WHICH START AT ADDRESS $0000

                        * FILE NAME: MEMCMD.SRC

                        * By Ron Drexler, 311 Wilson Way, Larkspur CA 94939

                        * DESIGNED TO ALLOW PAUSE ON OLD FORM FEED
                        * COMMANDS TO RUN IN MEMORY

                        * FLEX SYSTEM EQUATE
           C000 FLEX    EQU   $C000     USE $A000 FOR FLEX 2.0

                        * FLEX EQUATES
           CC12 USRCMD  EQU   FLEX+$0C12
           CC09 PSFLAG  EQU   FLEX+$0C09
           CD03 WARMS   EQU   FLEX+$0D03
           CCE4 POUT    EQU   FLEX+$0CE4 FLEX PRINTER OUTPUT
           CC1E TFRADR  EQU   FLEX+$0C1E TRANSFER ADDRESS

                        * EQUATES
           F3C0 RAMADR  EQU   $F3C0     MUST BE OUT OF FLEX'S WAY
           000C FORMF   EQU   $0C       FORM FEED

                        * VERSION CODE - OVERWRITTEN LATER
F3C0                    ORG   RAMADR
F3C0 0000               FDB   0         WASTE SPACE
F3C2 14            VER  FCB   20        VERSION 2.0

CC12                    ORG   USRCMD    SET MEMORY POINTER IN FLEX
CC12 F3C0               FDB   MEMTBL

F3C0                    ORG   RAMADR
F3C0 50 53         MEMTBL FCC   /PS/
F3C2 00                 FCB   0
F3C3 F3D7               FDB   PSOM
F3C5 46 46              FCC   /FF/
F3C7 00                 FCB   0
F3C8 F3D0               FDB   FEED
F3CA 4C 43              FCC   /LC/
F3CC 00                 FCB   0
F3CD F3DE               FDB   LSTCMD
F3CF 00                 FCB   0

F3D0 B6 000C       FEED LDAA  #FORMF
F3D2 BD CCE4            JSR   POUT
F3D5 20 1C              BRA   RETRN

F3D7 B6 FF         PSOM LDAA  #$FF
F3D9 B7 CC09            STAA  PSFLAG
F3DC 20 15              BRA   RETRN

F3DE BE CC1E       LSTCMD LDX   TFRADR    GET TRANSFER ADDR.
F3E1 26 0E              BNE   JUMP       IF NOT ADDRESS 0, GO TO IT

                        * IF ADDRESS 0, LOOK FOR STARTING PROGRAM JUMP
F3E3 A6 84              LDAA  0,X        GET FIRST BYTE OF PROGRAM
F3E5 81 7E              CMPA  #$7E       JMP?
F3E7 27 08              BEQ   JUMP       OK
F3E9 81 20              CMPA  #$20       BRA?
F3ED 27 04              BEQ   JUMP       OK
F3EF 81 16              CMPA  #$16       LBRA?
F3F1 26 02              BNE   RETRN
F3F3 6E 84         JUMP JMP   0,X

F3F5 7E CD03       RETRN JMP  WARMS

                        END

0 ERROR(S) DETECTED
```

SYMBOL TABLE:

```
FEED  F3D0  FLEX  C000  FORMF 000C  JUMP  F3F1  LSTCMD F3DE
MEMTBL F3C0 POUT  CCE4  PSFLAG CC09 PSOM  F3D7  RAMADR F3C0
RETRN  F3F5 TFRADR CC1E USRCMD CC12 VER   F3C2  WARMS  CD03
```

FORTH Interest Group

P.O. Box 8231 · San Jose, CA 95155 · (408) 277-0668

November 22, 1985

FOR IMMEDIATE RELEASE

Linda Kamo--Publicist
(213) 478-8703

OFFICIALLY ENDORSED FORTH STANDARD NOW AVAILABLE

ONLY FROM THE FORTH INTEREST GROUP

San Jose, CA, Nov. 22 -- The FORTH Interest Group (FIG) is pleased to announce the publication of the FORTH-83 Standard, the most recent Forth standard. This publication is the only one endorsed by Forth Standards Team, the organization responsible for its development. It is an authoritative description of the Forth-83 Standard and is designed to allow for portability of Forth-83 standard programs in source form between Forth-83 standard systems. Along with defining the terms and requirements of the standard, it addresses the following: references, compliance and labeling, usage, error conditions, glossary notation, required word set, double number extension word set, assembler extension word set, system extension word set and controlled reference words. It is available directly from the FORTH Interest Group for only $15.00.

The FORTH Interest Group (FIG), is a worldwide non-profit organization with over 5,000 members. It's 90 chapters are devoted to the Forth computer language. FIG membership of $20.00/yr ($33.00 foreign) includes a subscription to FORTH Dimensions, a bi-monthly publication. FIG also offers product discounts, group health insurance, an on-line data base, a job registry, and a large selection of Forth literature and many other services. For additional information contact the FIG HOT LINE (408) 277-0668 or write PO Box 8231, San Jose, CA 95155.

" " " " "

"Founded by & for FORTH users"

1208 NW Grant
Corvallis OR 97330
13 November 1985

Computer Publishing Center
68' Micro Journal
5900 Cassandra Smith Rd.
Hixson TN 37343

Dear Mr. Williams:

About a year ago I wrote an implementation of Forth-84 for 6809 systems with flex9. I recently converted the code for 6800 use, and I am releasing both versions for free distribution.

The implementation is quite complete, including a trace/debug function and a multitasker of the i/o-driven type, as well as the usual editor, assembler, etc. Anyone who would like a copy of the system, along with complete source code (in Forth) should send me two (2) 5-inch diskettes with a reusable mailer and return postage. Please be sure to indicate which cpu version is wanted.

Readers with access to Compuserve can also find the 6809 object code and limited documentation in the CLM SIG's DL7.

Yours truly,

Wilson M. Federici
1208 NW Grant
Corvallis OR 97330

alpha micro

CONTACT: Dennis Trombley
ALPHA MICRO
(714) 957-8500
(714) 641-6211
or
Brad Stevens
HILL AND KNOWLTON, INC.
(213) 937-7460

ALPHA MICRO INTRODUCES
NEW FAMILY OF 16/32-BIT MULTI-USER COMPUTERS

SANTA ANA, CA, December 16, 1985 -- Alpha Micro, a Santa Ana-based manufacturer of multi-user, multi-tasking computers is introducing the AM-1500 Series, its latest addition to the company's family of 16/32-bit microcomputer systems. The first two models in this new series, the AM-1545 and AM-1555, are based on the Motorola MC68010 microprocessor and industry standard VME bus architecture.

Offering increased performance, expandability and space-saving styling, the AM-1500 Series provides users with faster, more efficient processing power in a 12-60 user multi-tasking computer system, according to Len Palladino, vice president/general manager of Alpha Micro's Computer Systems Division.

"We designed the AM-1500 Series to work as a primary system for a small business or a departmental system for a large corporation," Palladino explained.

Key advantages of the AM-1500 Series, Palladino stated, are its increased processing power, expandability, upgradable configuration and compact size (20" wide x 7" high x 22½" deep). Perhaps the most important of these features is the ability to easily upgrade the system to a full 32-bit configuration.

"The industry standard VME bus used on the AM-1500 Series allows users to upgrade the system by installing a 68020 processor board," he said. "Thus, this series of computers will ultimately offer the processing speed and expanded user support of the 32-bit microprocessor."

The AM-1500 Series currently includes two models: the AM-1545 and AM-1555. The AM-1545's minimum configuration includes 2 MB of memory, 70 MB of hard disk and 12 I/O ports. The AM-1555 features 3 MB of memory, 140 MB of hard disk and 16 I/O ports. In addition, the complete systems (CPUs and terminals) can be connected to each other, as well as other Alpha Micro systems, to form local area networks (LANs) of multi-user, multi-tasking computer systems.

Both models in the AM-1500 Series can operate as either a desktop or a freestanding tower unit. The AM-1545 offers a choice of three hard disk backup options including floppy disk, streamer tape or Alpha Micro's proprietary videotape backup system. The AM-1555 is available with the videotape backup system only.

The AM-1545 retails for $22,915 to $26,585 depending on the configuration, and the AM-1555 ranges in price from $31,250 to $33,085. Delivery of both systems began in early November, 1985.

Founded in 1977, Alpha Micro provides computing and communications solutions to end users, system resellers and OEMs. The Computer Systems Division offers a family of high-performance, multi-user systems. The Video Technology Division markets Alpha Micro's patented video technology in the form of several unique data backup and distribution products, and the Service Division provides a comprehensive package of service and support options.

\# \# \#

microware
MICROWARE SYSTEMS CORPORATION

NEWS RELEASE

MICROWARE SYSTEMS CORPORATION
Drew Crane
515-224-1929

SUBJECT: MICROWARE ANNOUNCES POWERFUL NEW OS-9 NETWORKING

DES MOINES, IA -- Microware Systems Corporation has announced its new OS-9 Network File Manager (NFM), a powerful software-based networking system for computers based on the 68000 and 6809 processor families. The network software runs under Microware's popular real-time, multi-tasking OS-9 Operating System

The Network File Manager combines the file and input/output systems of all connected computers into one logical file system. Any network user can directly access files and I/O devices on any other system on the network as if they were local files. This makes the network system extremely easy to use. Also, existing utility application software can be used as is.

Because Network File Manager has a software-based architecture, it is completely hardware independent. It can be used with a wide variety of standard local area network or long-haul data communications hardware. The (NFM) can easily be customized for specific communications controllers using simple standard or user-written driver modules. It is compatible with virtually all popular standards such as ETHERNET, OMNINET, ARCNET, IEEE-488, etc. Multiple networks can be connected to a single system. This configuration allows direct inter-network access. Therefore "gateways" between different networks can be established.

Special security features built into the Network File Manager allow full control of local file access from other systems Combined with the operating system's standard file security mechanism, this provides complete file protection from unauthorized access.

The Network File Manager has undergone extensive testing at Beta test sites world wide, and is immediately available under OEM licenses or by single copy "NetPaks". A "NetPak" is a special package that allows a user to customize and install the network on their own systems. It includes the Network File Manager software, sample source code for typical device driver modules and special installation documentation.
Microware System Corporation is a Des Moines, Iowa-based company that specializes in operating systems and programming languages

for 68000 and 6809 family microprocessors. Founded in 1977, Microware is one of the oldest microprocessor system software houses in existance. Over 200 manufacturers have licensed Microware's OS-9 Operating System for a wide variety of applications such as communications products, industrial automation, instrumentation, personal computers, process control and more.

U s i n g F O R T H

Oswain Belgard
2419 Chartres
New Orleans, LA 70117
(504) 947-6227
CIS COCO SIG 75716,1124

I would like to add some to what Ron Anderson said about FORTH in his December 1985 column on languages. I'm by no means an expert, but I will try to provide FORTH approximations to the examples that were given for BASIC, C and PASCAL.

FORTH consists of functions or subroutines called words grouped into vocabularies, all of which together are called the dictionary. FORTH programming consists of extending the language by defining new words (in terms of words already defined) and adding them to the dictionary.

The FORTH code

```
7 EMIT
```

is equivalent to WRITE[CHR(7)); in PASCAL, putchar(7); in C, or PRINT CHR$(7) in BASIC. Note that FORTH uses postfix notation -- the argument precedes the word.

We can define a new word named print_uppercase as follows to show how EMIT could be used within a definition:

```
: print_uppercase     ( -- )   BL WORD 1+ C@
                      DUP 96 > OVER 124 < AND
                      IF 32 - EMIT
                         ELSE EMIT THEN ;
```

The colon is a defining word that creates the new word print_uppercase . The parentheses enclose a comment, in this case a simple stack diagram, which is ignored by the text interpreter. What a word expects on the stack is on the left side of the dash in the diagram and what it leaves on the stack is on the right side. A dash by itself means the word expects nothing and leaves nothing.

The word BL puts the ASCII code for a blank on the stack. The input stream is read by the next word, which is named WORD , using the blank as a delimiter. The delimiter value is consumed (removed from the stack) and the string is stored at an address named HERE -- the next free location above the dictionary.

The address HERE is put on the stack by WORD automatically in standard FORTH systems, but in some FIG-FORTHs such as STEARNS ELECTRONICS you will need to push it on the stack yourself by inserting the word HERE after WORD.

Since WORD puts a count in the first byte of the string, HERE must be incremented with 1+ to get the character address on the stack. C@ (pronounced "c-fetch") replaces the address with the ASCII code stored there. The word DUP duplicates this number on top of the stack (n -- n n) so the next word won't consume the original. OVER copies the second value on the stack to the top (n1 n2 -- n1 n2 n1).

The > (greater-than) and < (less-than) words leave flags on the stack to be logically AND-ed for the IF...ELSE...THEN structure that follows. The semi-colon ends the definition.

It's easy to "factor out" any segment of a definition as a seperate word to avoid repeating it in some other definition. We could factor print_uppercase as follows:

```
: getchar    ( -- char )   BL WORD 1+ C@ ;
: lowercase?   ( char -- char flag )
               DUP 96 > OVER 124 < AND ;
```

Then the new definition of print_uppercase would be:

```
: print_uppercase    ( -- )  getchar lowercase?
                     IF 32 - EMIT
                        ELSE EMIT THEN ;
```

To use this word you simply follow it with a character--

```
print_uppercase a
print_uppercase Z
```

--and it will output uppercase regardless of whether the input was upper or lower.

The more you factor out repetitious code, the more compact your program becomes. But this obviously works against speed since factoring out words requires additional instructions for branching, which slows the program down.

The main way to speed up FORTH is not to avoid factoring, however, but to use code words where time is critical. Such words are defined by the word CODE instead of the colon. Their definitions consist largely of words in the FORTH assembler vocabulary. Most of these look like the Assembly Language mnemonics of the host CPU.

The 1+ in the definitions above, for instance, could be defined as follows with the assembler vocabulary of Charles Eaker's eFORTH:

```
code  1+    ( n -- n+1 )    d pulu  1 # addd
            d pshu  next  end-code
```

The 6809 U register is used as the pointer to the eFORTH data stack. The definition pulls the top value on the stack into the D accumulator. It then adds one to the value in the accumulator and pushes the new number back on the stack. The word next is the address interpreter which gets FORTH from one word to another.

Seldom are programs written entirely in code words. One of the good things about FORTH is that you can mix high-level words with assembler words and colon-defined with code-defined words. They all use the same stack to pass parameters. This permits a trade off of execution speed for development speed or vice versa as needed.

Dear Don;

Since you published my FLEX HELP patch in your December issue, several people have called me to ask if I had a version for STAR-DOS. Seems more people are running STAR-DOS than FLEX on the PT-69 and other S8C's.

At the time, I hadn't, but in response to those requests, here it is. Unfortunately, the two are not interchangeable, but each version will perform identically with it's appropriate DOS.

HELP-SD LISTs a text file on the system drive called HELP.SYS when a question mark is input as a command.

This STAR-DOS version has a slightly more sophisticated drive number routine. It will find your system drive if it's not the customary drive 0.

Also, it has occurred to us that this routine could be used to output "tracking" help files, appropriate to wherever the operator was in a particular series of programs, simply by having those programs periodically overlay appropriate help file names over the "HELP.SYS" string in this patch as those programs progress.

HELP-SD.TXT is the assembler input listing. HELP-SD.ASM is the assembled output listing. HELP-SD.BIN is the assembled binary output file. HELP .SYS is the sample help text file.

Three cheers for STAR-DOS and that guy who invented it!

Jon H. Larimore
Editor's Note: Jon, thanks for the STAR-DOS version. I know that most of those using STAD-DOS thank you also.

I think you can understand the success of STAR-DOS is the fact that it is supported, and that makes a lot difference to any user or potential user.

Upwards, bigger and better are all fine aspirations, but if you ignore yuur foundation of support, eventually it catches up with you. STAR-DOS folks (Peter Stark, et al) seem to understand that. At least for now.

Too many counted FLEX down and out, too soon. But for a simple single-user system, it is yet to be topped. STAR-DOS seems to do everything that FLEX did, but somewhat better at times. It all adds up to the point of acceptance that STAR-DOS has today in ours and other market places. You would be surprised at the wide spread use that STAR-DOS has come into, in the short time span of its availability.

Let's hope they never forget.

```
                        ****************************************
                        * 6809 STAR-DOS HELP Patch             *
                        *                                      *
                        * By Jon H. Larimore                   *
                        *  5900 Arlington Blvd.                *
                        *  Arlington, Va. 22204                *
                        *                                      *
                        * Last edit 11/10/85                   *
                        *                                      *
                        * HELP is a patch to the STAR-DOS DOS which will
                        *  list a text file on the system drive called
                        *  "HELP.SYS" whenever a "?" is input as a
                        *  command. (Instead of simply outputing
                        *  "ERROR 26".) It is intended to make STAR-DOS
                        *  somewhat friendlier for new or casual users.
                        *
                        * HELP will temporarily activate the TTYSET
                        *  depth and pause controls, then return
                        *  them to their previous settings.
                        *
                        * HELP normally resides in the unused STAR-DOS
                        *  print spooler area in a PT-69's RAM.
                        *  If you're using this area for a spooler or
                        *  other routine however, you may wish to ORG
                        *  it at high memory (resetting MEMEND), or
                        *  perhaps in another available area.
                        *
                        * You may either append this to your terminal
                        *  drivers while creating a new STAR-DOS.SYS,
                        *  or simply SET it using the STARTUP routine.
                        *
                        * The "ldxerr" equate below is an undocumented
                        *  location in 6809 STAR-DOS Version 1.4.  If
                        *  you're using a different version of STAR-DOS,
                        *  you should verify it's accuracy prior to
                        *  assembly.
                        *
                        * To verify this, use your monitor memory dump
                        *  routine to look at address $D11B.  If the
                        *  code "8E C982" is there, you're OK.  If not,
                        *  then again use the memory dump routine to find
                        *  the string "(hex 0D 0A) ERROR", (located at
                        *  $C9B1 in 6809 STAR-DOS Version 1.4).  This is
                        *  your "errstr" equate.  Then, find the "8E nnnn"
                        *  (LDX #$nnnn) code in which "nnnn" is the
                        *  address of the first byte of "ERROR" in your
                        *  STAR-DOS.  The address of the "8E" byte of that
                        *  "8E nnnn" code should be your correct
                        *  "ldxerr" equate.
                        *
                        * Also, check the value of "tradsp" here.  It is
                        *  set for a 24-line terminal display.  You may
                        *  need to change it.
                        * Note that this number must be ONE GREATER than
                        *  your terminal can display.


                        * DOS Constants
                        *
C080  linbuf  equ   $c080     Input line buffer
C700  pspool  equ   $c700     Print spooler area (unused?)
C71B  quecnt  equ   $c71b     Print queue counter
CC03  depcnt  equ   $cc03     TTY display depth count
CC09  paucnt  equ   $cc09     TTY pause control
CC0B  sysdno  equ   $cc0b     TTY system drive number
CC11  lsttrm  equ   $cc11     Last terminator character
CC14  bufpnt  equ   $cc14     Input line buffer pointer
CC16  escreg  equ   $cc16     Escape return register
CD03  warmst  equ   $cd03     Warm start
CD1E  pstrng  equ   $cd1e     Print a string
CD4B  dscmd   equ   $cd4b     Call DOS as a subroutine

C9B2  errstr  equ   $c9b2     >>> VERIFY BEFORE ASSEMBLY <<<
D11B  ldxerr  equ   $d11b     >>> VERIFY BEFORE ASSEMBLY <<<


                        * If "?" is input as a DOS command, then
                        *  list "HELP.SYS"

D11B                    org   ldxerr    Point to DOS "ERROR" process area
D11B 7E                 fcb   $7e       Stuff "JMP" mnemonic into this byte
D11C C71F               fdb   helpi     Stuff address into jump vector

                        * Run this routine in the unused spooler area
                        *
C700                    org   pspool

                        * Delete from here to "tradsp" if you relocate
                        *  this code to another memory area
                        *
                        * Disable print spooler jump vectors
                        *
C700 39 39 39           fcb   $39,$39,$39 Put RTS's here
C703 39 39 39           fcb   $39,$39,$39
C706 39 39 39           fcb   $39,$39,$39
C709 39 39 39           fcb   $39,$39,$39
C70C 39 39 39           fcb   $39,$39,$39
C70F 39 39 39           fcb   $39,$39,$39
```

```
99
100  C718                       org    qucnt    Spooler queue counter
101  C718 00                    fcb    0        Force queue count to 0
102
103  C71A 19            trmdsp  fcb    25       Your terminal's lines +1?
104  C71B 00            dspflg  fcb    0        Temporary depth flag
105  C71C 00            pauflg  fcb    0        Temporary pause flag
106
107  C71D 84  CC1B  help1   lda    lstlrm   Get the last non-alpha character input
108  C722 81  3F            cmpa   #'?      Was it a question mark?
109  C724 26  50            bne    help9    No, output "ERROR 26" etc.
110  C726 B6  CC0D          lda    sysdrv   Yes, get system drive number
111  C729 8B  30            adda   #$30    Convert it to ASCII
112  C72B B7  C70E          sta    sysdrv   Store it
113  C72E 10CE C739         lds    #hlpstr  Yes, point to HELP file name
114  C732 8E  C09B          lds    #lnebuf  Point to first character of input buffer
115  C735 A6  A0    help2   lda    ,r+      Get a character, point to the next one
116  C737 81  04            cmpa   #$04     Is it the string terminator?
117  C739 27  05            beq    help3    Yes
118  C73B A7  80            sta    ,r+      No, store it, point to next buffer location
119  C73D 7E  C735          jmp    help2    Store next character
120  C740 CC  C09B  help3   ldd    #lnebuf  Get input buffer address
121  C743 FD  CC14          std    bufptr   Store it in buffer pointer
122                 *
123                 * Set TTY terminal display lines and pause
124                 *
125  C746 7D  CCD3          tst    depcnt   Is terminal line depth set?
126  C749 26  07            bne    help4    Yes
127  C74B 7C  C71B          inc    deptlc   No, set temporary flag
128  C74E B6  C71C          lda    trmdsp   Get number of lines terminal can display
129  C751 B7  CCD3          sta    depcnt   Store it in TTYSET depth register
130  C754 7D  CC09  help4   tst    pauctn   Is Pause on?
131  C757 26  08            bne    help5    Yes
132  C759 7C  C71E          inc    pauflg   No, set temporary flag
133  C75C 86  FF            lda    #$FF     Get "pause on" value
134  C75E B7  CC09          sta    pauctn   Turn terminal "pause" on
135  C761 CE  C764  help5   ldu    #help6   Get return address for escape-return
136  C764 FD  CC16          std    escrtn   Store it
137  C767 B6  CD4B          jsr    docmd    Call DOS as a subroutine
138                 *
139                 * Reset TTY values to previous settings
140                 *
141  C76A 7D  C71B  help6   tst    deptlg   Temporary depth setting?
142  C76D 27  06            beq    help7    No
143  C76F 7F  C71B          clr    deptlg   Yes, clear flag
144  C772 7F  CCD3          clr    depcnt   And clear depth counter
145  C775 7D  C71E  help7   tst    pauflg   Temporary pause setting?
146  C778 27  04            beq    help8    No
147  C77A 7F  C71E          clr    pauflg   Yes, clear flag
148  C77D 7F  CC09          clr    pauctn   And clear pause control
149  C780 7E  CD03  help8   jmp    warmst   Return to warm start
150                 *
151  C783 8E  E402  help9   ldx    #errstr  Load "ERROR"
152  C786 7E  D11E          jmp    IOutPre3 And continue DOS error routine
153
154  C789 4C 49 53 54  hlpstr  fcc  'LIST,'
155  C78D 30           sysdrv  fcc  '0'      System drive number (ASCII)
156  C78F 2E 48 45 4C          fcc  '.HELP.SYS',ID,4
157
```


Dear Don,

First off, I must apologise to my 6800 friends for brushing them off the way I did in my last letter. Most of what I said applies equally well to 6800 systems, of course, except for the two sections relating to the display of error-messages and the LISTing of .BAC programs.

So ... by way of peace offering, I went back through my original disassembly of FLEX2's XBASIC, and located the equivalent areas to be modified. Here they are :

1. To LIST .BAC files, locate the following code in the vicinity of address $099B -- 7D 011C 27 05 86 40, and change the 7D to 39, using your system Monitor. Otherwise, everything is as it was for 6809.
2. To display error-messages, locate the following somewhere near address $0BC3 :

Present code	Change to
CE 0C0F	B7 AC20
BD 05B4	8C AC1F
CE 0122	BD AD3F
5F	5F
F7 0126	F7 0126
BD 0FC6	BD AD24

In addition, the expanded ERRORS.SYS (containing the complete list of error messages) will be required. Be warned that SOME error-code numbers have been changed since FLEX2 days, but it should be a simple matter to adjust the latest 6809 list appropriately.

Having got that off my chest, I want to say a little about "hacking". In the October issue Ron Anderson handed out a little bouquet to those of us who try to help newcomers to the 68xx scene who are experiencing difficulties in getting their systems functional. This is most likely because we all remember our early days, and the struggles we had, WITH NO KNOWLEDGEABLE PERSON AROUND TO HELP US. There was only one way out of our difficulties then and that was to fumble and bumble around, and read what scant literature was available, until eventually a little light dawned - or we gave up in frustration (for the time being, at least). Ron, you omitted mentioning one of the most active sources of help and encouragement over the years - namely one Ronald W. Anderson. The reason you know of OUR activities is because you, too, are right in there corresponding with the self-same people. So, a big "Thank you, Ron" on behalf of all of us!!
Now let's get on with LESSON 4 of our XBASIC series.

Last time I mentioned some errors in the XBASIC manual. I have also located an error in XBASIC itself, which I doubt will ever get fixed by TSC now that they've abandoned FLEX. Maybe it's already shown up in some of your programs, and left you a little puzzled. What happens is that XBASIC will report Error #55 (Unbalanced parentheses) in Line xxxx, yet a listing of that line will show that such is not the case. In actual fact, it should have reported Error #78 (Undimensioned Array reference) and normally occurs when a DIM statement has been omitted. It only seems to occur when an array-dimension is out of range in a doubly-dimensioned array. To make this perfectly clear, let's assume the following :

```
50 DIM A(6,2)
60 PRINT A(7,1)
```

RUNning this program will cause a correct Error #77 (Array reference out of range), but if Line 50 is deleted, it will come up with an incorrect "Error #55 (Unbalanced Parentheses) in Line 60". This had me very puzzled the first time it occurred as (wouldn't you know it?) that particular line seemed to have about 20 sets of parentheses in it. I think that what occurs is that when XBASIC checks the first dimension and finds that it's out of range, it abandons the whole process and doesn't continue on to check for the matching right-paren, and thus reports an Error #55. This is supported by the fact that if you simply ask it to PRINT A(7) it responds with the correct error-message.

Let's move on to something else .... namely the CHAIN instruction, which is normally only invoked when we have a huge program which won't fit into memory, and yet is structured in such a way that when one section has been executed, we have no further need of it. We simply move into the second half of our program, and NEVER again go back to the first. In such a case, I split my program into two sections, <game.BAS>, let's say, and <game.NXT>. I like to keep the name of the game intact, and change the extension only, so that both sections will be listed together in my master Directory of all disks. Herein lies a problem!!!

Earlier versions of XBASIC, when executing the CHAIN instruction, made the assumption that if a CHAINed file didn't have a .BAC extension it was assumed to be

a .BAS type of file and was loaded as such. No difficulty with those versions. Somewhere along the way, however, newer versions reversed this assumption (i.e. if the extension wasn't .BAS then it had to be .BAC and was loaded as a .BAC file). Under the later conditions, the first half of a .BAS program would execute OK, but not the second (extension = .NXT) -- unless I gave the CHAINed file the extension .BAS as well.
What to do? Obviously, I had to jump in and modify XBASIC (later versions only - but I'm afraid I don't know at which version the change-over was made). Anyway, the change is quite simple. Somewhere around address $2FA8 locate the code :

```
42 41 26 04 A6 0E 81 53 10 26 D9 5F
 B  A              S
```

and change to :

```
42 41 26 07 A6 0E 81 43 10 27 D9 5F
 B  A              C
```

All for now! Maybe next time I'll get around to the cause of this mini-tutorial on XBASIC, namely the addition of an EDIT command to the language. Well, it wasn't really a tutorial, but a discussion of things you won't find in the Manuals. See you next month.

Sincerely,

R. Jones
President

Winterthur, 07 december 1985

Werner Thie
Siedenstr 18
8400 Winterthur
Switzerland

68 MICRO JOURNAL
5900 Cassandra Smith Rd
Hixson, Tn 37343
USA

Software order for FLEX on 5 1/4" Disks single sided single density

Dear Sirs

I order the following items from you

MJ Disk-23 ISAM Condon airmail delivery	$ 15.55
3 year subscription surface to 68 MJ	$100.50
Total	$116.05

PS: Is there a known patch to the following problem with TSC debug: executing a PULS PC works as it should. But it should also decrement the subroutine level count, which it does definitely not. So it is likely to run into the error message: nesting too deep. The problem is extremely boring when debugging C programs, specially recursive ones. If there is help, please let me know.

Kind regards
and keep on going
we count on you

Dimensional Software
7704 Pickard Ave. NE
Albuquerque, NM 87110
505-835-1516

Dear Larry,
I thought I'd drop you a line and let you know how much I've been enjoying my subscription to your fine publication. It's nice to see the new typeset format. I've been aware of your magazine for quite a while. In my association with J & M Systems, Ltd. in Albuquerque, and now consulting independently, I've had opportunity to work with several 68XXX systems (from an old SWTP 6800 to new Cimix and Smoke Signal machines).

As you may know, J & M supplies the CoCo market with disk controllers and drives. While CoCo magazines are okay for the begining, low budget hacker, the authors are often misguided (everyhody is patching everything !) and the ads heavily sales oriented (everything's on sale!). I find you publication much more "meaty" and the authors well informed.

Attending school here at New Mexico Tech in Socorro, I've been programming "C" on the VAX and appreciate the flexablity it offers over other languages. My adviaor claims that a properly equiped 68020 system can out perform a VAX 11/750. Quite a claim, but still believable. I expect to see "C" become the language of choice for many new and old microcomputer users. The "C" User Notes in your magazine will help support and introduce prosepctive programmers to this expressive, coming language.

Like some other magazines I receive, I enjoy looking at the ads as much as reading the articles! Your magazine is a wealth of information and a great shopping guide for systems designers. I sincerely thank you for your generosity.

Lastly, some more comments...I'd like to see the ISAM routines in "C" and some more information about compiler implementations on 68XXX computers. It may lead to smaller, better implementations. Also, some articles on the new 68000 machines coming out (Amiga and that Atari, for instance) might expose some untapped markets.

Thanks again,

Bruce Amos

Scintillex Software

Dear UniFLEX System Manager,

Scintillex Software has an exciting new UniFLEX program ready for you: a full-featured, screen-oriented calculator. Now before you say, "Big deal,..I already have a calculator", take a look at the features of this package. First of all, it has the ability to work in hexadecimal, binary, and octal, as well as decimal. Wouldn't your programmers love that? It also simulates a paper tape on the screen an sends the tape data to a file for subsequent hardcopy. It has a ASCII code converter that will convert a character to a value or a value to a character at the press of a button. It has ten memory registers. It performs logical operations (like And, Or, and Exclusive Or). It has on-screen help.

I could go on and on listing features, but you can read them for yourself in the enclosed flyer. This program is so easy to use that you won't need the manual after one quick glance. And the way it draws the calculator and paper tape on the screen actually makes it fun to use.

Scintillex Software has other UniFLEX packages available too. Many of you are already owners of the **Initializer II** and the **Disk Maintenance Package**. Those of you who aren't, don't know what you are missing! You may not have a need for the Disk Maintenance Package, but every UniFLEX system can benefit from the capabilities of the Initializer II. Whether it be to automate your system, to make it more reliable, to make it useable by novices, to better control a modem, or to prevent unauthorized use, the Initializer II is a real plus for any UniFLEX system. Sound like magic? It isn't. It's just a good idea that have been well developed to give you control over the way your system boots and runs. Let us know if you would like more information on what the Initializer II (or the Disk Maintenance Package) can do for you.

Sincerely yours,

Daniel E. Vanado
President, Scintillex Software

Don Williams Sr.
68 Micro Journal - CPI
5900 Cassandra Smith Drive
Hixson, TN 37343

October 30, 1985

Dear Don,

We today received a call for technical assistance from a user of INTROL-C who very recently purchased a copy of our C6809-OS9O9 compiler package through Southeast Media and were a little suprised to learn he had obtained it for use on a Level I OS9 host. Please advise your sales personnel at Southeast Media that any sales of OS9 versions of INTROL-C should be strictly limited to OS-9 Level II users only; we do NOT recommend the compiler for use on the Level I systems, nor have we recommended it for Level I hosts since introducing our Version 1.50 software package in the Summer of 1984.

With the increased size of the Version 1.50 and later releases of the INTROL-C package - coupled with the relatively large amount of memory consumed by the Level I operating system itself - there simply is not enough free memory left for the compiler to run comfortably on a 64K system running Level I. Because of this, we strongly discourage its use under Level I.

John Wisialowski

Frank Black Miller, M.D., P.A.
Adult, Adolescent, Child and Family Psychiatry
Central Medical Park, #502
2609 N. Duke Street
Durham, N.C. 27704

Dear Mr. Williams:

I am responding to your "Ramblings and Such" column in the current issue of 68' Micro Journal.

Let me give you a portrait of myself. I am a psychiatriat. I got started in computers 5 years ago by buying a used Radio Shack Model I because I heard it did "electronic typing" that meant you could type something and revise as often as you wanted without paying a typist. When I bought the thing I didn't realize I needed a printer. I didn't know about disk drives, memory size, nothing...It took me over a year to get an expansion interface with the 48K, a disk drive, and a printer. I learned what to use R.S. for and what not to. I bought Electric Pencil, got dissatisfied and started buying DOS's and word processor through the mail and didn't get burned once, ended up with Lazewriter and Zorlof, both absolutely superb programs, and I was on my way. I had an old LP II, then bought an MX-80 through the mail through a liquidation sale (maybe it was a "hot" machine, who knows). Then I got a second full Model I system and another printer and had a system at home and at the office. Then I started getting interested in databases since my billing system was only a repetitive word processor based contrivance, of manually calling up files, typing in changed dates, printing out bills. No accounting, no ageing of accounts, eta. Databases were bad for the Model I. It was a dinosaur when I bought it, as the Model III and then IV took over. I couldn't change over every year and blow $2,000 just to keep up...However,I found that medical billing programs, and databases for the Model I were simply terrible and inadequate.

Then one of my I's started to act like a I, glitchy, restarts, overheating, etc. I tolerated it for two years, cleaning contacts, all the usual superstitious Model I maintenance. Finally one of them gave up the ghost a year ago. I atarted looking around for an alternative. The Commodore 64 was a junk machine, so were all the Atari's, etc. The IBM had taken the world by storm but I couldn't afford the $3,000 for a PC and I got further turned off by the average price of software, $400 for a word processor that was clunky (WordStar) and more for databases. Secondly, I read that the 8086-8 wasn't God's gift to computer caaes....Apple was a kiddie machine and the Lisa-Mac mess was just that. I agonized thinking I had to be MS-DOS compatible. Then I started reading about UNIX and knew this was the DOS of the future, forget 8086....however, I could not afford a BIG machine either to run UNIX. Being an inveterate Radio Shacker, I don't know why...I was drawn to the Color Computer. When I saw the price, and heard about OS-9, I was hooked. Now I have three of the creatures, one of them for a daughter. I have dispersed, donated and junked my I's, and converted to the CoCo. I have started toying with OS-9 and FLEX. I have a RS-DOS databased billing system (runs on Pro-Color-File) that is a dream. My secretary can run incredibly simple word processors on it and do fine, and I can run BIGTIME stuff like Stylograph. And the software prices are STILL reasonable. And I still have one at home, another at the office. A computer addict's dream.

So why do I read 68' Micro. I can't understand one-tenth of it. I don't know Assembly Language at all, but I am starting...I have only recently started to do more with OS-9 than load and run a program. I have bought BASIC-09 and now finally I am motivated to learn BASIC--I have to confess, I had only learned a bit of it before...I have bought the Rainbow Guide to OS-9 and will purchase your-all's books on OS-9 and Flex. Very simply put, I read everthing I can get on the Color Compter and I was switched to your journal from Color Computer Journal. I love your magazine. It motivates me to learn more. I am now convinced more than ever that the 68XX and 68XXX is where it will be as an alternative to Intel's 8086etc chips. I will someday have something approaching UNIX.

Sincerely yours,
Frank Miller
Frank Black Miller, M.D.

AMERICA ADO, INC

Contact: Brian Ogihara, ADO (213) 532-5010
Tom Brigham, B/S, Inc. (213) 550-7145

Release Date: November 18, 1985

MULTI-USER, MULTI-TASKING 68010 MICROPROCESSOR-BASED CPU MULTIBUS BOARD INTRODUCED BY AMERICA ADO

TORRANCE, CA -- America ADO, Inc., has introduced its FA*STAR C681-01 Integrated Board Computer utilizing a 16-bit 68010 virtual memory microprocessor operating at 8, 10 or 12.5 MHz plus an on-board 68451 memory management unit providing multi-user and multi-tasking support. The new board is compatible with IEEE-796 and iLBX bus standards.

"The introduction of the FA*STAR C681-01 multi-user, multi-tasking CPU further broadens our Multibus CPU line providing our customers with even more capabilities" explained Brian Ogihara, General Manager of America ADO. "By offering such a wide range of Multibus boards, users realize the time, cost and delivery benefits of dealing with one source."

Features of the FA*STAR C680-01 include 64K byte SRAM memory with dual-port function which is expandable to 128k with an expansion board, 4 JEDEC 28-pin sockets, 16-bit timer, Centronics parallel port and 2-channel RS-232C serial port. There are 14 interrupt levels expandable to 71 by cascading. As well all of the company's products, the board is covered by America ADO's two-year warranty.

In addition to Integrated Board Computers, America ADO also markets a complete line of IEEE-796 compatible boards, including an intelligent disk controller, a LAN controller, a graphic display controller and a variety of additional memory Multibus boards.

America ADO is located at 1840 West 186th Street, Suite 200, Torrance, California 90504. Phone is (213) 532-5010.

PLEXUS

Brenda Birrell
Plexus Computers, Inc.
(408) 943-2248

PLEXUS P/35 AND P/60 UPGRADEABLE TO 68020 CPU

SAN JOSE, Calif., Nov. 6, 1985 -- Plexus Computers, Inc., has introduced a job processor field upgrade kit for its P/35 and P/60 UNIX-based computer systems. The kit will upgrade the system's job processor from a Motorola 68000 to a 68020, thereby increasing processing speed from 1.5 to 2.0 times, depending on the application.

The upgrade kit also contains an 8 Kb high-speed memory cache and optional floating point support based on the Motorola 6881 co-processor.

Plexus's 68020-based board will be available for field upgrade in the first quarter of 1986, priced at $8,000, with a $1,000 discount for the return of the old board. The Motorola 68881 floating point co-processor, priced at $1,500, will be available first quarter, 1986.

Policy of upgrading continues

The new upgrade kit reflects Plexus's policy of enhancing earlier products with the latest technology. In the past five years, the company's previous upgrades have quadrupled the capacity of its original system.

"Our strategy is to provide systems with the longest life cycle for the lowest cost in the marketplace," said Kip Myers, vice president of marketing at Plexus. "We do this by producing inexpensive but powerful upgrades as an alternative to system replacement."

Plexus recently announced upgrades for the P/60 that provide up to 16 Mb of main memory, up to 1.2 gigabytes of disk storage, and faster I/O for storage peripherals. The P/35 computer can now be fitted with up to 8 Mb of main memory.

The P/35 and P/60 computers feature multiprocessor architectures optimized for the UNIX System V Release 2.0 operating system. Both machines offer cache memory, a high-speed memory map, extensive self-diagnostics, and LSI circuitry to minimize component failure.

Plexus Computers, Inc., manufactures a fully compatible line of high-performance 32-bit computer sytems designed for the VAR/OEM and volume end-user. Plexus computers utilize the UNIX operating system for commercial applications and employ a unique multiprocessor architecture which provides users with exceptional power and performance. Plexus markets its products through sales offices in the U.S. and via distributors and subsidiaries throughout the world.

For further information, contact Brenda Birrell, marketing communications manager, Plexus Computer, Inc., 3833 North First Street, San Jose, CA 95134, (408) 943-2248.

MOTOROLA
Semiconductor Products Inc.
P.O. BOX 20912 PHOENIX, ARIZONA 85036

EDITORIAL CONTACT
Ed Prestwood
(602) 994-6959

READER CONTACT
Microsystems Marketing Dept.
(602) 438-3501

MOTOROLA OFFERS MAP KIT TO SPEED DEVELOPMENT OF FACTORY AUTOMATION

PHOENIX, AZ, NOVEMBER 4, 1985... Developers of local area network applications incorporating the Manufacturing Automation Protocol (MAP) can now shorten their development cycle by using the MAP Developer's Kit just introduced by Motorola. The kit provides VMEbus-compatible 10 Mb/sec MAP network interface boards, software, and complete broadband network hardware, including cabling.

The MAP Network interface board sets supplied with the Kit were recently introduced by Motorola with associated resident network software implementing layers of 2-4 of the seven-layer Open System Interconnect model of the ISO and MAP 2.1 committees.

Network developers will find the kit useful not only as a MAP design-in tool for the engineering lab, but also as part of a final installation package for a factory or office environment. All network cabling supplied in the kit is constructed of materials rated by Underwrites Laboratories for application in a test environment or in a permanent installation.

The Developer's Kit provides all network elements required to implement a complete operating 10 Mb/sec two-node or four-node network, except for the host systems into which the VMEbus network node cards are plugged. A VMEbus-compatible host may be a specialized unit provided by the developer; or alternately, it may be a standard system package available from Motorola.

A variety of useful support software is also supplied in the Developer's Kit in UNIX System V format, on 5 1/4" diskette. Operating network software implements MAP 2.1 layers through layer 4, downloadable into the MAP network interface from the user-provided host.

To futher assist the developer, both source and object versions of demonstration software are included for verifying that the assembled MAP network is functioning properly. Software interfaces and buffer management techniques used in the demonstration software serve as models, to accelerate development of the final application programs.

Contacts: Jo Ann McDonald, PR Counsel
415/964-7423
or: Wayne Fischer
Director of Marketing
408/354-3410

FORCE COMPUTERS, INC.
727 University Ave.
Los Gatos, California 95030

FORCE COMPUTER INTRODUCES FASTEST 32-BIT CPU for VMEbus

Los Catos, CA, October 9, 1985....Force Computers' most significant introduction of the year, the CPU-20/21 is the first 32-bit VMEbus product series on the market to fully maximize the power of the high performance 68020 microprocessor, running a average of 50% faster than competitive boards. The CPU-20 (and CPU-21 with 68881 floating point co-processor) performs at 16.7 MHz, becoming the only 68020-based CPU boards to run without wait states, while executing 32-bit accesses to as much as 1 MByte of high-speed local static RAM.

According to Wayne Fischer, Director of Marketing at Force Computers Inc., "We were not the first VMEbus supplier with 68020 boards, but we're definitely surpassing all competition with our 1 MByte no wait state execution. Our selection of a 32-bit datapath for the EPROM area maximizes execution speed for real time processing, compared to the 16-bit local data bus used on early 68020 CPU boards."

Of particular interest, the CPU-20/21 features 2 serial I/O ports: one RS232 compatible for use as a debug port for the monitor or as the master terminal (supervisor), the second (RS232 or RS422 compatible via jumper setting) is a general purpose interface for printer or host connection, making the CPU-20/21 readibly usable as a stand-alone computer. The other leading 68020 product on the current market requires a special board for RS232 drivers/receivers to interface to interface to a terminal, occupying one additional slot.

Another major feature setting the CPU-20/21 apart from the competition is the use of Force's Local Memory Expansion (FLME) which provides the only fast and simple implementation to extend local memory currently available on the market. "This is a must for those who need to manipulate large amounts of data at very high speeds.

Basically it's the next best thing to a native bus," says Flacher.

PLME is an interboard bus that is physically connected by a P3 connector utilizing the same connector type as used for VMEbus P1 and P2....effectively putting the functionality of three boards onto one board. PLME provides memory expansion without relying on a slower local bus like VMX or MVMX32, eliminating two or three wait states from the most common operations without increasing the load on the VMEbus.

Ideal application areas for the CPU-20/21 include: real time process and vision control, signal processing, advanced factory automation, laser printing, tomography, geophysics and seismographics, nuclear reaearch and radar applications.

Within the product series, the fully configured CPU-21 includes the 68881 Floating Point Coprocessor running at 16.7 MHz, up to 512 Kbytes of 32-bit wide EPROM, a VMXbus interface and two multiprotocol aerial I/O channels. Under software control, these channels can be configured for a variety of standard serial protocols, such as asynchronous, SDLC, HDLC, and X.25 operation. At no extra cost the CPU-21 offers twice the data throughput from onboard EPROM relative to other 68020 designs.

Key features of the CPU-20/21 include:

* 68020 CPU Operating at 16.7 Mhz
* 68881 Floating Point Coprocessor
  Operating at 16.7 MHz (only CPU-21)
* Full VMEbus Interface:
  A32, A24, A16 Address Width
  D32, D24, D16, D8 Data Width
  Interrupt Handler and
  Single Level Arbiter
* 256 KBytes to 1 MByte of No Wait State Static RAM
* Full VMXbus Primary Master Interface Including Interrupt Handler
* 4 EPROM Sockets (32 Bit Wide)
* 4 EPROM/SRAM Sockets (32 Bit Wide)
  up to 512 Kbytes of On-Board EPROM Space
  up to 128 KBytes of On-Board SRAM Space
* 2 Serial I/O Interfaces with Multiprotocol Communicationa Controller Support
  1 RS232 Compatible Interface
  1 RS232/RS422 Interface (Selectable)

Pricing for the CPU-20/21 series starts at $5,450. Delivery is 30 days ARO from Force Computers' facilities in Los Gatos, California; Munich, West Germany; or Paris, France. In comparison to the competition, 'The CPU-20/21 is the Ferarri of VMEbus: not inexpensive, but sleek and screaming fast...it's what price-to-performance is all about. The CPU-20/21 product aeries marks a new plateau in VMEbus progress," concludes Wayne Flacher, who is also chairman of the IEEE Standards Committee on VMEbus.

Rapidly growing Force Computers Inc. is now the number one independent supplier of VMEbus products and systems worldwide. The company is noted for its broad base of VMEbus products that clearly offer customers the best price to performance ratios on the market. Force is a multinational corporation with headquarters in Los Gatos, California and subsidiaries in Munich, West Germany and Paris, France.

FORCE'S CPU-5 PRODUCES FASTEST 68000 PERFORMANCE FOR VMEbus SYSTEMS AT 16.7 Mhz.

Los Gaeos, CA, November 6, 1985.....Force Computers today announced the fastest 68000 based CPU on VMEbus currently available...the CPU-5. The new CPU card utilizes a 12.5 or 16.7 Mhz 68000 16-bit microprocessor ana 128K (to 512K) bytes of zero-wait-state static RAM to provide high speed program execution. Plus, the powerful 68881 floating point co-processor is available to enhance arithmetic operations. System bus speed is further increased by providing a 68450 DMA controller.

In addition, two multiprotocol serial I/O channels, 256 Kbyte EPROM capacity, a single level VMEbus arbiter, a parallel interface/timer, and a full VMXbus interface make the CPU-5 truly the most powerful and flexible 16-bit VMEbus multiprocesaing engine available.

According to Jim Green, Manager of Applications Engineering at Force Computers, "Incorporating this large number of VLSI devices on a single VMEbus board has allowed us to take major system functions and tie them very tightly together. This idea, along with putting most board parameters under software control, has defined a new generation of bus structured products that ease system integration and increase overall system performance."

CPU-5's VMXbus interface conforms to Revision B of the specification and provides high speed access to memory and peripheral I/O devices. This feature improvea system speed by reducing traffic on the VMEbus.

A 68230 Parallel Interface/Time (PI/T) provides software control over such function as interupt request levels, bus arbitration, bus release parameters, and status displays. Each of the 68561 MPCC serial chips provide programmable interupt levels and vectors, and are connected with on-board support devices.

Jim Green also points out that, because of its speed, the CPU-5 will be especially useful in applications requiring high numerical computation capability, primarily critical real time process control or data analyais.

Featurea of the CPU-5 include:

- 68000 microprocessor with 12.5 or 16.67 MHz clock
- or 68010 microproceasor with 12.5 MHz clock
- 68881 floating point co-processor at 12.5 or 16.7 Mhz
- 68450 four channel DMA controller
- 128 or 512 Kbytes static RAM providing zero wait state access up to 16.67 MHz
- 256 Kbytes EPROM capacity (using 27512 devices)
- Two 68561 multi-protocol communication controllers with on-board driver/reciever circuitry.
- 68230 PI/T for local software control
- Single level VMEbus arbiter
- VMXbus interface

Introductory pricing for both the 68000 and 68010 versions of the CPU-5's begin at $2,495, quantity one. Delivery is 30 days ARO from Force Computers' facilities in Los Gatos, California; Munich, West Germany and Paris, France.

Rapidly growing Force Computera Inc. is now the number one independent supplier of VMEbus products and systems worldwide. The company is noted for its broad base of VMEbus products that offer customers the beat price to performance ratios on the market. Force is a multinational corporation with headquarters in Loa Gatoa, California and aubaidiaries in Munich, West Germany and Paris, France.

Classified Advertising

Tano Qutpoat II, 56K, 2 5"DSDD Drives, FLEX, MUMPS $895.
MICROKEY $4500 Single Board Computer, Target 128K RAM, FLEX, FORTH, with optional 6502 CPU & ROMS as advertiaed on p. 51 DEC. 84 68' Micro Journal. $2300.
LSI 68008 CPU Card with Digital Research CPM/68K $350.

1-PT-69 complete with Dual 5" DSDD Disk System and Controller, includes FLEX DOS.
TELETYPE Model 43 PRINTER - with serial (RS232) interface and full ASCII keyboard. LIKE NEW - new cost $1295.00 - ONLY $559.00 ready to run.
S/09 with Motorola 128K RAM, 1-MPS2, 1-Parallel Port, MP-09 CPU Card $1990. 1 DMAF2 Dual 8" Drives with Controller $2190. 1-CDS1 20 Meg Hard Disk System with Controller $2400.

I will accept any reasonable counter-offer !!

Call Tom (615) 842-4600 M-F 9 A.M. to 5 P.M. E.S.T.

***

Wanted - empty SWTP 6800 cabinet, with or w/o power supply. Joe Ponder, 1245 Weiland #3, Kent, WA. 98031.

***

COMPUTER/COMPONENTS HELIX 6809 SS50C BUS CHASSIS, PS, MB, 2 PORTS; 6809 CPU; 8/9 DMA CNTLR; 64K STATIC RAM; OS9/L1; OS9/L2; FLEX 09; HELIXBUG; 5" USSD 40 & DSDD 80 TRK CIMIX 6800 2MHZ CPU; 16K RAM; SSB BFD SD DISK CNTRLR; SWTPC CHASSIS; 6800 1MHZ CPU 8K RAM NO CHIPS; FLEX OS, UTIL, DIAG, SORT. CASH, VISA, MASTER, AMER EXP. ARRANGEMENT GEORGE INGRAM (312) EVE 653-0360 DAY 531-4607.

***

GMX mainframe, SWTP MP-09, 56K, 2 SSSD 5 1/4" drives, ADM-5 terminal, Gimini 10 printer, OS-9 v1.1, BASIC09, ASM., EDIT, DEBUG $2000. Will consider any reasonable offer. G.H. Holloman, Box 79129, Jackson, Ms. 39236.

***

Versabus system - includes VM01A1 CPU, VM11-2 512K RAM, VM21 disk controller, VM30 communication module, 13 Meg hard disk, 16 slot card cage with backplanes, power supplies, cables, etc. $1500. Hugh Shane, PO Box 45292, Seattle, Wa. 98145 (206) 523-4235.

***

I would be interested in getting in touch with other 68' readers/users in the Netherlands. Could you be of help? Berend A.S. Storm, Voortstukken 20, 9761 JT EELDE, the Netherlands.

WINDRUSH MICRO SYSTEMS

UPROM II

PROGRAMS and VERIFIES: I2758, I2508, I2716, I2516, I2732/2732A, MCA68764/6, I2766/2766A, I2564, I27128/27128A, and I27256. I=Intel, T=Texas, M=Motorola.

NO PERSONALITY MODULES REQUIRED!

TRI-VOLT EPROMS ARE NOT SUPPORTED

INTEL's intelligent programming (tm) implemented for Intel 2764, 27128 and 27256 devices. Intelligent programming reduces the average programming time of a 2764 from 7 minutes to 1 minute 15 seconds (under FLEX) with greatly improved reliability.

fully enclosed pod with 5' of flat ribbon cable for connection to the host computer MC6821 PIA interface board.

MC6809 software for FLEX and OS9 (Level 1 or 2, Version 1.2).

BINARY DISK FILE offset loader supplied with FLEX, MDOS and OS9.

Menu driven software provides the following facilities:

a. FILL ..... a selected area of the buffer with a HEX char.
b. MOVE ..... blocks of data.
c. DUMP ..... the buffer in HEX and ASCII.
d. FIND ..... a string of bytes in the buffer.
e. EXAMINE/CHANGE . the contents of the buffer.
f. CRC ..... checksum a selected area of the buffer.
g. COPY ..... a selected area of an EPROM into the buffer.
h. VERIFY ..... a selected area of an EPROM against the buffer.
i. PROGRAM ..... a selected area of an EPROM with data in the buffer.
j. SELECT ..... a new EPROM type (return to types menu).
k. ENTER ..... the system monitor.
l. RETURN ..... to the operating system.
l. EXECUTE ..... any DOS utility (only in FLEX and OS9 versions).

FLEX AND OS9 VERSIONS AVAILABLE FROM GIMIX. SSB/MDOS CONTACT US DIRECT.

PL/9

- Friendly inter-active environment where you have INSTANT access to the Editor, the Compiler, and the Trace-Debugger, which, amongst other things, can single step the program a SOURCE line at a time. You also have direct access to any FLEX utility and your system monitor.
- 375+ page manual organized as a tutorial with plenty of examples.
- Fast SINGLE PASS compiler produces 8K of COMPACT and FAST 6809 machine code output per minute with no run-time overheads or licence fees.
- Fully compatible with TSC text editor format disk files.
- Signed and unsigned BYTES and INTEGERS, 32-bit floating point REALs.
- Vectors (single dimension arrays) and pointers are supported.
- Mathematical expressions: (+), (-), (*), (/), modulus (\), negation (-)
- Expression evaluators: (=), (<>), (<), (>), (>=), (<=)
- Bit operators: (AND), (OR), (EOR/XOR), (NOT), (SHIFT), (SWAP)
- Logical operators: (.AND), (.OR), (.EOR/XOR)
- Control statements: IF..THEN..ELSE, IF..CASE1..CASE2..ELSE, BEGIN..END, WHILE.., REPEAT..UNTIL, REPEAT..FOREVER, CALL, JUMP, RETURN, BREAK, GOTO.
- DIrect access to (ACCA), (ACCB), (ACCD), (XREG), (CCR) and (STACK).
- FULLY supports the MC6809 RESET, NMI, FIRQ, IRQ, SWI, SWI2, and SWI3 vectors. Writing a self-starting (from power-up) program that uses ANY, or ALL, of the MC6809 interrupts is an absolute snap!
- Machine code may be embedded in the program via the 'GEN' statement. This enables you to code critical routines in assembly language and embed them in the PL/9 program (see 'MACE' for details).
- Procedures may be passed and may return variables. This makes the functions which behave as though they were an integral part of PL/9.
- Several fully documented library procedure modules are supplied: IOSUBS, BITIO, HARDIO, HEXIO, FLEXIO, SCIPACK, STRSUBS, BASTRING, and REALCON.

'... THIS IS THE MOST EFFICIENT COMPILER I HAVE FOUND TO DATE.'

Quoted from Ron Andersons FLEX User Notes column in '68. Need we say more?

MACE/XMACE/ASM05

All of these products feature a highly productive environment where the editor and the assembler reside in memory together. Gone are the days of tedius disk load and save operations while you are debugging your code.

- Friendly inter-active environment where you have instant access to the Editor and the Assembler, FLEX utilities and your system monitor.
- MACE can also produce ASMPROCs (GEN statements) for PL/9 with the assembly language source passed to the output as comments.
- XMACE is a cross assembler for the 6800/1/2/3/8 and supports the extended mnemonics of the 6303.
- ASM05 is a cross assembler for the 6805.

D-BUG

LOOKING for a single step tracer and mini in-line disassembler that is easy to use?? Look no further, you have found it. This package is ideal for those small assembly language program debugging sessions. D-BUG occupies less than 0c (including its stack and variables) and may be loaded anywhere in memory. All you do is LOAD IT, AIM IT and GO! (80 col VDUs only).

McCOSH 'C'

This is as complete a 'C' compiler as you will find on any operating system for the 6809. It is completely compatible with UNIX V7 and only lacks 'bit-fields' (which are of little practical use in an 8-bit world!).

- Produces very efficient assembly language source output with the 'C' source optionally interleaved as comments.
- Built-in optimizer will shorten object code by about 11%.
- Supports interleaved assembly language programs.
- INCLUDES its own assembler. The TSC relocating assembler is only required if you want to generate your own libraries.
- The pre-processor, compiler, optimizer, assembler and loader all run independantly or under the 'CC' executive. 'CC' makes compiling a program to executable object as simple as typing in 'CC,HELLO.C <RETURN>'.

IEEE-488

- SUPPORTS ALL PRINCIPAL MODES OF THE IEEE-488 (1975/8) BUS SPECIFICATION:
  - Talker - Serial Poll - Single or Dual Primary Address
  - Listener - Parallel Poll - Secondary Address
  - System Controller - Group Trigger - Talk only ... Listen only
- Fully documented with a complete reprint of the KILOBAUD article on the IEEE bus and the Motorola publication 'Getting aboard the IEEE Bus'.
- Low level assembly language drivers suitable for 6800, 6801, 6802, 6803, 6806 and 6809 are supplied in the form of listings. A complete back to back test program is also supplied in the form of a listing. These drivers have been extensively tested and are GUARANTEED to work.
- Single S-30 board (4, 8 or 16 addresses per port), fully socketed, gold plated bus connectors and IEEE interface cable assembly.

PRICES

D-BUG	(6809 FLEX only)	$ 75.00
MACE	(6809 FLEX only)	$ 75.00
XMACE	(6809 FLEX only)	$ 98.00
ASM05	(6809 FLEX only)	$ 98.00
PL/9	(6809 FLEX only)	$198.00
'C'	(6809 FLEX only)	$295.00
IEEE-488	with IEEE-488 cable assembly	$298.00
UPROM-II/U	with one version of software (no cable or interface)	$395.00
UPROM-II/C	as above but complete with cable and S-30 interface	$545.00
CABLE	5' twist-n-flat 50 way cable with IDC connectors	$ 35.00
S-30 INT	S-30 interface for UPROM-II	$130.00
EXOM INT	Motorola EXORbus (EXORciser) interface for UPROM-II	$195.00
UPROM SFT	Software drivers for 2nd operating system. Specify FLEX or OS9 AND disk size!	$ 35.00
UPROM SRC	Assembly language source (contact us direct)	

ALL PRICES INCLUDE AIR MAIL POSTAGE

Terms: CWO. Payment by Int'l Money Order, VISA or MASTER-CARD also accepted.

WORSTEAD LABORATORIES, NORTH WALSHAM, NORFOLK, ENGLAND. NR28 9SA.

TEL: 44 (692) 404086
TLX: 975546 WMICRO G

WE STOCK THE FOLLOWING COMPANIES PRODUCTS: GIMIX, SSB, FHL, MICROWARE, TSC, LUCIDATA, LLOYD I/O, & ALFORD & ASSOCIATES.

FLEX (tm) is a trademark of Technical Systems Consultants, OS-9 (tm) is a trademark of Microware Systems Corporation, MDOS (tm) and EXORciser (tm) are trademarks of Motorola Incorporated.

ALL SYSTEMS INCLUDE:

- The CLASSY CHASSIS with a ferro-resonant, constant voltage power supply that provides + 8 volts at 30 Amps, + 16 volts at 5 Amps, and − 16 volts at 5 Amps.
- Gold plated bus connectors.
- Double density DMA floppy disk controllers.
- Complete hardware and software documentation.
- Necessary cables, filler plates.

YOU CAN EXPAND YOUR SYSTEM WITH:

MASS STORAGE

Dual 8" DS00 Floppies, Cabinet & Power Supply $1898.88
60MB Streamer (UniFLEX-020 only) $2400.00
1.6MB Dual Speed Floppy ..... (under development)

MEMORY

#67 Static RAM - 64K NMOS (6809 Only) $349.67
#64 Static RAM - 64K CMOS w/battery (6809 Only) $398.64
#72 256K CMOS Static RAM w/battery $998.72
#31 16 Socket PROM/ROM/RAM Board (6809 only) $268.31

INTELLIGENT I/O PROCESSOR BOARDS

significantly reduce systems overhead by handling routine I/O functions; freeing the host CPU for running user programs. This improves overall system performance and allows user terminals to be run at up to 19.2K baud. For use with GMX III and 020 systems.

#11 3 Port Serial-30 Pin (OS9) $498.11
#14 3 Port Serial-30 Pin (UniFLEX) $498.14
#12 Parallel-50 Pin (UniFLEX-020) $538.12
#13 4 Port Serial-50 Pin (OS9 & UniFLEX-020) $618.13

I/O BOARDS (6809 SYSTEMS ONLY)

#41 Serial, 1 Port $88.41
#43 Serial, 2 Port $128.43
#46 Serial, 8 Port (OS9/FLEX only) $318.46
#42 Parallel, 2 Port $88.42
#44 Parallel, 2 Port (Centronics pinout) $128.44
#45 Parallel, 8 Port (OS9/FLEX only) $198.45

CABLES FOR I/O BOARDS—SPECIFY BOARD

#95 Cable sets (1 needed per port) $24.95
#51 Cent. B.P. Cable for #12 & #44 ..... $34.51
#53 Cent. Cable Set $36.53

OTHER BOARDS

#66 Prototyping Board-50 Pin $56.66
#33 Prototyping Board-30 Pin $38.33
Windrush EPROM Programmer S30 (OS9/FLEX 6809 only) $545.00

CONTACT GIMIX FOR FURTHER DETAILS ON THESE AND OTHER BOARDS AND OPTIONS.

EXPORT MODELS: ADD $30 FOR 50Hz. POWER SUPPLIES.
ALL PRICES ARE F.O.B. CHICAGO.

GIMIX DOES NOT GUARANTEE PERFORMANCE OF ANY GIMIX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.

GIMIX, Inc. reserves the right to change pricing, terms, and products specifications at any time without further notice.

GIMIX 2MHZ 6809 SYSTEMS

GMX 68020 SYSTEMS

Operating Systems Included	#49 OS9 GMX I/ and FLEX	#39 OS9 GMX II/ and FLEX	#79 OS9 GMX III/ and FLEX	#39 UniFLEX	#89 UniFLEX III	#020 UniFLEX VM
CPU included	#05	#05	GMX III	#05	GMX III	GMX 020 + MMU
Serial Ports Included	2	2	3 Intelligent	2	3 Intelligent	3 Intelligent
High Speed Static RAM	64KB	256KB	256KB	256KB	256KB	1 Megabyte
PRICES OF SYSTEMS WITH: Dual 80 Track DS00 Drives	$2998.49	$3398.39	$4898.79	N/A	N/A	N/A
25MB Hard Disk and one 80 track Floppy Disk	$5598.49	$5998.39	$7798.79	$5998.39	$8098.89	$13,680.20
72 MB Hard Disk and one 80 track	$7598.49	$7998.39	$9798.79	$7998.39	$10,098.89	$15,680.20
a 72MB + a 6MB removable pack hard disk and one 80 track floppy	$9098.49	$9498.30	N/A	$9498.39	N/A	N/A
a 72MB + a 12MB removable pack hard disk and one 80 track floppy	N/A	N/A	$11,298.79	N/A	$11,598.89	$17,180.20
GMX 6809 OS9/ FLEX SYSTEMS SOFTWARE OS9 + Editor, Assembler, Debugger	GMX I Included	GMX II Included	GMX III Included			
FLEX	Included	Included	Included			
GMXBUG Monitor	Included	Included	Included			
Basic 09, Run8 (OS9)	Included	Included	Included			
RMS (OS9)	Included	Included	Included			
DO (OS9)	Included	Included	Included			
VDisk for FLEX	N/A	Included	Included			
RAMDisk for OS9	N/A	$125 option	Included			
0-FLEX	N/A	$250 option	Included			
Support ROM	N/A	N/A	Included			
Hardware CRC	N/A	N/A	Included			

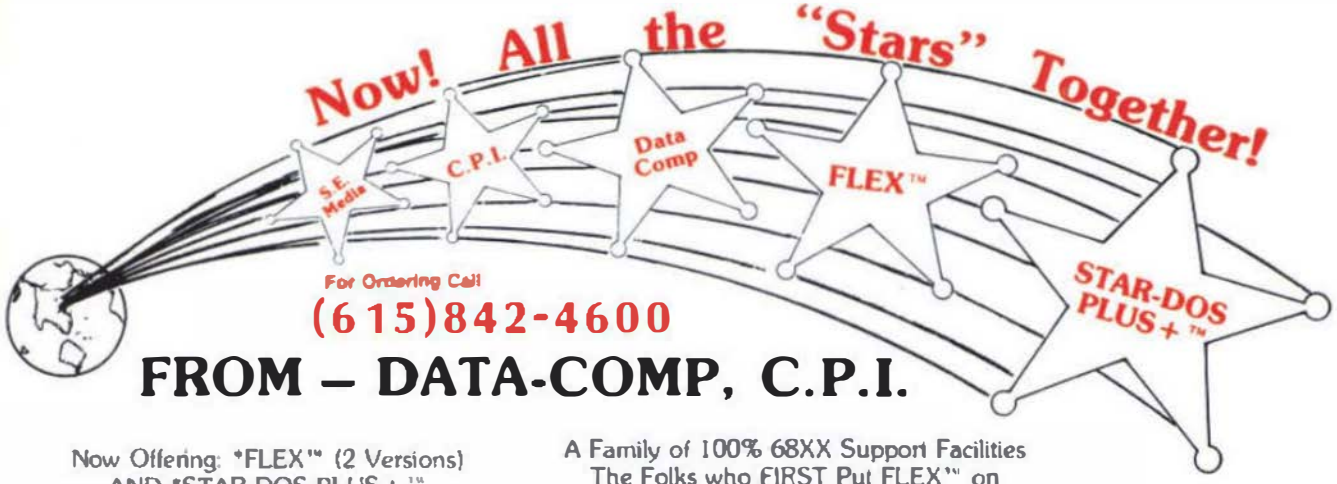
Available: Wide variety of languages and other software for use with either OS-9 or FLEX.

All GIMIX versions of OS9 can read and write RS color computer format OS9 disks, as well as the Microware/GIMIX standard format.

All OS9/FLEX systems allow you to software select either operating system.

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if order is under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account number 73-32033.

GIMIX inc.
1337 WEST 37th PLACE
CHICAGO, ILLINOIS 60609
(312) 927-5510 • TWX 910-221-4055



For Ordering Call
(615)842-4600

FROM – DATA-COMP, C.P.I.

Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00
Only $79.95

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.95

FLEX-CoCo Jr.
without TSC Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utilities
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program
+ Many Many More!!!

TSC Editor
Reg $50.00
NOW $35.00

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Systems

2 THINLINE DOUBLE SIDED DOUBLE DENSITY DISK DRIVES SYSTEM WITH POWER SUPPLY,CABINET,DISK DRIVE CABLE,J&M NEW DISK CONTROLLER JPD-CP WITH J-DOS,RS-DOS OPERATING SYSTEMS.	$469.95
* Specify What CONTROLLER You Want J&M, or RADIO SHACK	
THINLINE DOUBLE SIDED DOUBLE DENSITY 40 TRACKS	$129.95

Verbatim Diskettes

Single Sided Double Density	$ 24.00
Double Sided Double Density	$ 24.00

Controllers

J&M JPD-CP WITH J-DOS	$139.95
WITH J-DOS, RS-DOS	$159.95
RADIO SHACK 1.1	$134.95
RADIO SHACK Disk CONTROLLER 1.1	$134.95

Disk Drive Cables

Cable for One Drive	$ 19.95
Cable for Two Drives	$ 24.95

MISC

64K UPGRADE FOR C,U,E,F, AND COCO II	$ 29.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET FOR A SINGLE DRIVE	$ 49.95
DISK DRIVE CABINET FOR TWO THINLINE DRIVES	$ 69.95

PRINTERS

EPSON LX-80	$289.95
EPSON MX-70	$325.95
EPSON MX-100	$495.95

ACCESSORIES FOR EPSON

8148 2K SERIAL BOARD	$ 89.95
8149 32K EXPAND TO 128K	$169.95
EPSON MX-RX-80 RIBBONS	$ 7.95
EPSON LX-80 RIBBONS	$ 5.95
TRACTOR UNITS FOR LX-80	$ 39.95
CABLES & OTHER INTERFACES CALL FOR PRICING	

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

VISA

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
Telex 5106006630

Introducing

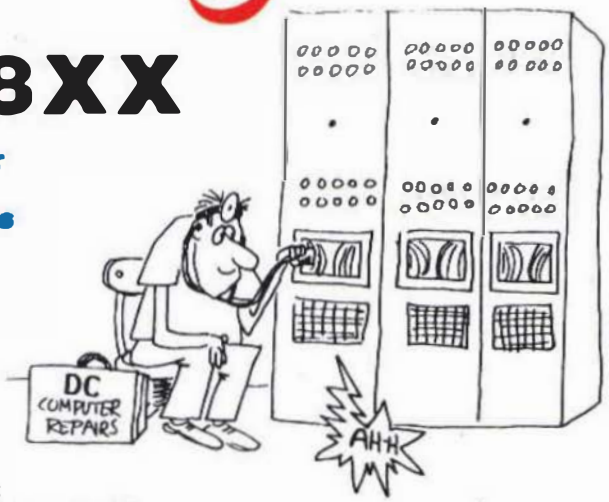
S-50 BUS/68XX

Board and/or Computer

Terminals-CRTs-Printers

Disk Drives-etc.

REPAIRS



This

Not This

NOW AVAILABLE TO ALL S50/68XX USERS

The Data-Comp Division of CPI is proud to announce the availability of their service department facilities to 'ALL' S50 Bus and 68XX users. Including all brands, SWTPC - GIMIX - SSB - HELIX and others, including the single board computers. * Please note that kit-built components are a special case, and will be handled on an individual basis, if accepted.

1. If you require service, the first thing you need to do is call the number below and describe your problem and confirm a Data-Comp service & shipping number! This is very important, Data-Comp will not accept or repair items not displaying this number! Also we cannot advise or help you troubleshoot on the telephone, we can give you a shipping number, but NO advice! Sorry!

2. All service shipments must include both a minimum $40.00 estimate/repair charge and pre-paid return shipping charges (should be same amount you pay to ship to Data-Comp).

3. If you desire a telephone estimate after your repair item is received, include an additional $5.00 to cover long distance charges. Otherwise an estimate will be mailed to you, if you requested an estimate. Estimates must be requested. Mailed estimates slow down the process considerably. However, if repairs are not desired, after the estimate is given, the $40.00 shall constitute the estimate charge, and the item(s) will be returned unrepaired providing sufficient return shipping charges were included with item to be serviced. Please note that estimates are given in dollar amounts only.

4. Data-Comp service is the oldest and most experienced general S50/68XX service department in the world. We have over $100,000.00 in parts in stock. We have the most complete set of service documents for the various S50/68XX systems of anyone - YET, WE DO NOT HAVE EVERYTHING! But we sure have more than anyone else. We repair about 90% of all items we receive. Call for additional information or shipping instructions.

DATA-COMP
5900 Cassandra Smith Rd
Hixson, TN 37343

☎
(615)842-4607
Telex 5106006630

MasterCard VISA